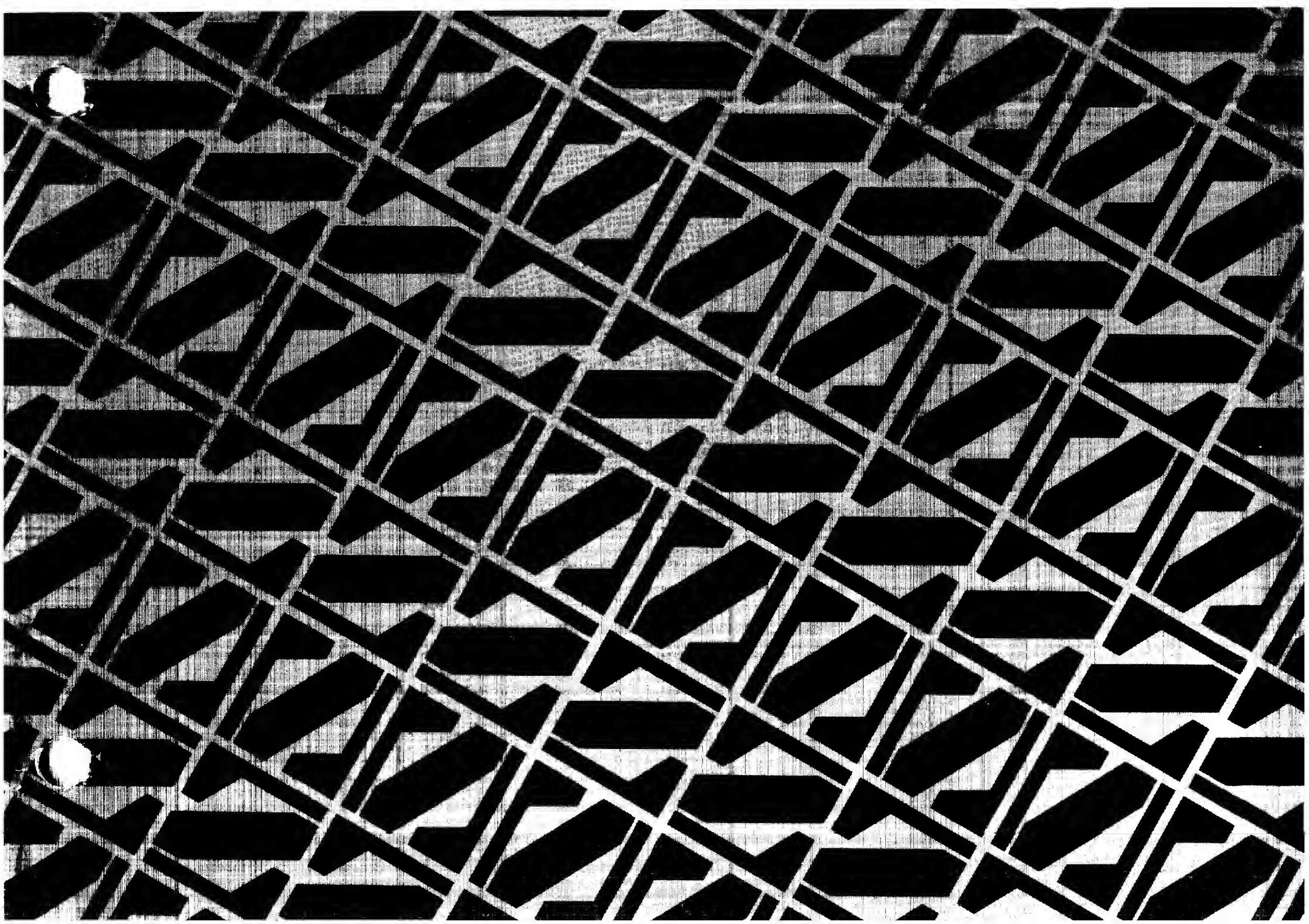
National Semiconductor

Order No. IMP-16S/025YB

Pub. No. 4200025B

IMP-16 Utilities Reference Manual



Order No. IMP-16S/025YB
Publication No. 4200025B

INTEGRATED MICROPROCESSOR

IMP-16
UTILITIES REFERENCE MANUAL

March 1974

2900 Semiconductor Drive
Santa Clara, California 95051

PREFACE

This publication provides user information about the IMP-16 Debugging Program (DEBUG), the Loader Programs, Teletype Input/Output Routines, Firmware (PROM, ROM) Paper Tape Generation, and the IMP-16 Source File Editor. Supplemental information is included in the appendixes. Familiarity with the IMP-16 Programming and Assembler Manual is a prerequisite for using and understanding the information described in this manual.

This manual was previously titled, "IMP-16L Utilities Reference Manual, Publications Number 4200025A". Henceforth, it shall be titled, "IMP-16 Utilities Reference Manual, Order No. IMP-16S/025YB and Publications Number 4200025B".

Copies of this publication and other National Semiconductor publications may be obtained from the sales offices listed on the back cover. The information in this publication is subject to change without notice.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

ILLUSTRATIONS

TABLES

Chapter 1

DEBUG

1.1 INTRODUCTION

DEBUG is a relocatable object program that supervises the operation of a user's program during checkout. DEBUG provides the following facilities for testing computer programs.

- Printing selected areas of memory in hexadecimal or ASCII format.
- Modifying the contents of selected areas in memory.
- Modifying computer registers including the stack.
- Inserting instruction breakpoint halts.
- Taking memory snapshots during execution of user's program.
- Initiating execution at any point in program.
- Searching memory.

1.2 USAGE

DEBUG is a relocatable load module that is loaded into the user's environment as any other relocatable program. DEBUG is initially entered through the global location DEBUG. This may be performed by the loader at the completion of the loading of all programs, by a direct jump from a user's program or by an alteration to the program counter at the control panel. DEBUG may also be entered through the global location, DEBUG1. This entry point loads locations 0 and 3 with an initialization routine that enables the user to recover to DEBUG by pressing the INITIALIZE button and then the RUN button.

DEBUG is self-contained; it includes all of its own input and output. The output and formatting ability of DEBUG is available to the user by subroutine calls to the global UCALL program in DEBUG. The calling sequence is as follows:

```
JSR      UCALL
.WORD    XXXXXXXX       ;BASE LOCATION TO PRINT
.WORD    XXXXXXXX       ;TOP LOCATION TO PRINT
```

NOTE

The definitions of the above assembler language instructions are given in the IMP-16 Programming and Assembler Manual. The locations specified are formatted with eight words per output line and are displayed on the Teletype. The use of UCALL is an independent subfunction of DEBUG and does not affect normal DEBUG control.

1.2.1 Memory Requirements

The following memory is needed to run DEBUG:

Top Sector: X'42D or 1069_{10} words.

Base Page: 6 words.

1.3 DEBUG LANGUAGE

The control statements which are used to command the operation of DEBUG are confined within a set of rules which define the syntax (the format of control statements) and semantics (the meanings of the various symbols and characters comprising the control statement) of the language.

1.3.1 Conventions Used In This Chapter

The following notation conventions are used to describe the commands, both in the general case (in the command descriptions) and in the specific case (in the examples).

- Mixed upper- and lower-case characters are used for comments and notes.
- Nonunderlined characters, numbers, and symbols, used in the examples, indicate computer-generated output from the Teletype printer.

  For example: TURN ON PUNCH

- Underlined characters, numbers, and symbols, used in the examples, indicate user-generated input at the Teletype keyboard. Two types of underlined statements are used:

  1) Lower-case statements or statement parts represent the general case (to be further defined by the rules of syntax).

  2) Upper-case statements or statement parts represent the exact (specific) form of the input required to be typed in.

  For example:

-T <address argument>	(general case)
-T 2345:7F	(specific case)
-NOTE ADDRESS	(specific case)

- Circled, upper-case characters represent operation of Teletype keyboard keys that do not generate a printed character.

  For example: CR represents the carriage return key.

  ALT MODE represents the alternate mode key.

- CTRL/x represents the operation of the CONTROL key in conjunction with an alphabetic key. The CONTROL key must be depressed first, prior to pressing the alphabetic key, and held in while the alphabetic key is pressed. The combined symbol is circled because a control operation is initiated; no printed symbol is produced.

1.3.2 Syntax and Semantics

The basic elements of DEBUG commands are defined below. In the formal (symbolic) descriptions of DEBUG commands, the following symbols are used:

<a>	Specifies an element "a" either of a command or of another element.
::=	May be read as, is defined as, and appears in a statement which defines the element to its left.

a | b | c	Indicates that at least one of the elements (a, b, c, etc.) should be specified in the command (a or b or c or ... etc.).
{a}	Indicates that one of the elements specified inside the braces must be included in the statement.
[a]	Indicates that the element(s) specified within the brackets are optional and need not be included in the command, unless desired.

1.3.3 Syntax

The following meanings are assigned to the terms used in the general-case form of the statements.

<hexadecimal number> ::=	From one to four digits from the hexadecimal set 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Leading zeros may be omitted.
<address argument> ::=	{ Memory address Memory address range Register address Register address range }
<memory address> ::=	A four-digit hexadecimal number specifying a memory location.
<memory address range> ::=	A memory address, followed with a colon (:), followed with a second memory address. For example: 3528:354A The memory address to the left of the colon represents the lower limit of the range; the memory address to the right of the colon represents the high limit of the range.
<register address> ::=	The letter 'R', followed with a two-digit hexadecimal number from the range of 0 through X'13. Because leading zeros may be omitted, the number may appear as a one-, or two-digit number. For example: R0, R13, R5
<register address range> ::=	The letter 'R' alone, specifying all registers; or a register address, followed with a colon (:), followed with a two-digit hexadecimal number. The register address to the left of the colon represents the lower limit of the range; the hexadecimal number to the right of the colon represents the upper limit of the range. The upper limit must be numerically greater than the lower limit. For example: R6:13
<comment> ::=	English language text, including letters and numbers, exactly as typed in.

<value> ::=	A four-digit hexadecimal number used as the contents of a memory location or the contents of a register. Consists of a 16-bit number. May also be specified as two ASCII characters. For example: 124A, 2235, 0060, 'GO'
ASCII characters ::=	A two-character set of ASCII characters, preceded with and terminated by a single quote. For example: 'AX'

1.3.4 Semantics

All numbers input to DEBUG are interpreted as hexadecimal digits. Characters may be either decoded as ASCII or hexadecimal, depending on the particular context. The following description explains the use of certain characters:

, (comma)	Delimiter of address and range arguments.
: (colon)	Delimiter for a range argument; signifies that all the locations from the first entry through the last are included in the range. For example, a:b signifies all the locations from a through b, including a and b.
R	Signifies that the address following the character R is a register address or register address range. A two-digit hexadecimal number is associated with the range symbol; this number is limited to the range 0 through X'13. The values 0 through 3 are for registers AC0, AC1, AC2, and AC3. The values 4 through X'13 are for the stack contents. R without any value represents all registers, 0 through X'13.
. (period)	The current location (CL) is the last location addressed by the previous DEBUG command.

1.4 COMMUNICATIONS

The user can communicate with DEBUG through a Teletype keyboard, printer, paper tape punch, and paper tape reader. Whenever DEBUG takes control, it types the minus sign character (-) to indicate that it is ready to accept a command. The user then may type control statements to direct the operation of DEBUG. All commands must be terminated by a carriage return CR or a line feed LF . To ignore a command, the ALT MODE key may be pressed at any time before the LF or the CR ; in this case, the # symbol is printed and no action occurs. If DEBUG detects an error in the command, it types a question mark (?) and prompts for a new command.

As information is transmitted to the Teletype, the Teletype is interrogated for input. If any character is detected, the current output is terminated and the user is prompted for another DEBUG command. This feature is particularly useful for terminating excessive or undesirable output.

More specific information pertaining to particular commands and situations is given in the appropriate command descriptions, below.

Control is returned to DEBUG from a user's program by use of the HALT command. Upon halting, DEBUG types the halt address. Control is transferred back to the user's program from DEBUG by the GO directive. Details pertaining to the HALT and GO directives are described under the descriptions of the commands themselves.

1.5 DEBUG COMMANDS

DEBUG commands consist of a single alphabetic character representing the command to be performed, followed by a list of the arguments for the commanded operation. The arguments are separated by commas. The numeric fields in an argument list must be in hexadecimal notation; leading zeros may be omitted. Blank characters are ignored, except when enclosed by quotation marks. In this manner, blanks may be used to enhance readability. A statement must be terminated by a carriage return or a line feed.

1.5.1 Command Descriptions

The commands that are used in DEBUG are described in the following paragraphs. In the examples contained within the command descriptions, all information input by the user is underlined to distinguish it from the information generated by DEBUG. This is fully discussed in paragraph 1.3.1.

1.5.2 TYPE

-T <address argument> [, <address argument> ......]

The contents of the specified locations are printed on the terminal in hexadecimal notation. The starting address is printed, followed by one to eight locations per line. If the contents of consecutive locations (starting at any location on a line and extending to the end of the line) are alike, only the first of the like locations is printed. The remaining ones are omitted. The next line also is omitted if the contents of all locations on the line are identical to the contents of the last printed location on the previous line. However, if the contents of any location within such a line are different, then the contents of the locations for that line are printed according to the rules given above. The address for a new line always is a multiple of 8 for consistency and ease of reading. The final location referenced becomes the value of the current location. The format for the TYPE printout is illustrated below.

A printout of the contents of locations 104 through 120 is requested as follows:

```
-T104:120 (CR)
```

DEBUG responds with the following output: (typical data)

```
0104  88DF  08DF  08DF  08FF  00FF  80FF  08FF  88FF
010C  00FF  88FF  08FF  88DF  88FF  805F  88DF  08DF
0114  08FF  08FF  88FF  08FB  88FB  80EB  80FB         <—— (Would have been 80FB)
011C  00FB  00FB  A8FB  88F3  4000
-
```

1.5.3 CHAR

-C <address argument> [, <address argument> ......]

This command performs in a manner similar to that of the TYPE command with the exception that the information is printed in ASCII character format. The output line consists of the contents of one to eight words preceded by the address of the first word. The format for the CHAR printout is illustrated below.

The following example shows the memory contents in locations 220 through 223, interpreted as hexadecimal (TYPE statement) and ASCII (CHAR statement).

Using the TYPE statement,

```
-T 220:223 CR

0220  4131  4242  4147  4243
-
```

Using the CHAR statement,

```
-C 220:223 CR

0220  A1  BB  AG  BC
-
```

An ASCII to hexadecimal conversion table is included in appendix A.

1.5.4 ALTER

A { <memory address> | <register address> } , <hexadecimal number> [, <...> , ...]

The ALTER command alters the contents of the specified memory or register location(s). Arithmetic and stack registers have register addresses 0 through 3 and 4 through X'13, respectively. Address 4 is the top of the stack; that is, the last value placed in the stack. The user specifies the address of the location or register that is to be altered, followed by a comma and the value to be placed in the addressed location. If the value is terminated by CR , the value is stored and the ALTER is terminated. If an LF terminates the expression, the value is stored and the user is prompted with the address of the next location or register. A comma may be used to indicate successive locations or registers to be altered without repeated prompts. Any error detected by DEBUG during processing prevents the alteration of the location being processed. Pressing the ALT MODE key terminates the ALTER without storing the last value. The value of the last prompt is the new current location.

The following example describes altering locations 2212 through 2215:

```
-A 2212, 0000, 0CDD, 00F7, 80EB CR
```

The following example describes the use of the LF key and the CR key to control and terminate an ALTER operation:

```
-A12A, 5133 LF
012B  8 LF
012C  'AA' CR
-
```

In the above example, after altering location 012A, DEBUG responds with the address of the next location to be altered (012B). Without further initialization, the value 8 is entered. The second LF causes DEBUG to again prompt with the next available location to be altered (012C). The ASCII characters AA are entered and the ALTER operation is then terminated with a CR . The final prompt (-) signals the user that DEBUG is ready for another command input.

1.5.5 HALT

-H <memory address> [, <hexadecimal number>]

HALT terminates control by the user's program at the location specified by the memory address and returns control to DEBUG. The hexadecimal number (optional), if specified, is the number of times to pass through this location before halting execution. The instruction at the given location is not executed when execution of the user's program is terminated; the instruction is normally executed immediately after control is returned to the user's program by use of the GO command.

The HALT command can be used successfully only when the instruction at the HALT location and the instruction at the following location always are executed consecutively. Thus, the instruction at a HALT location cannot be either a skip or a transfer such that the instruction at the following location would not be executed consecutively. Execution of the HALT does not remove it from the user's environment. It is in effect each time that the instruction at the specified location is executed.

NOTE

> DEBUG allows the user to set up to seven HALTS and/or SNAPs. If the user attempts to specify an eighth HALT or SNAP, DEBUG responds with an OV and refuses to accept the command.

All registers are saved by DEBUG when it is entered and restored upon a GO command. The register stack must have one location for the DEBUG HALT execution. Upon halting, DEBUG prints CLxxxx, where xxxx is the location of the halt.

For example, a breakpoint halt is required, in the user's program, at location 200. However, if it is desired to halt only after the fifth pass past that point in the routine, the command may appear as follows:

```
-H 200,5 (CR)
```

1.5.6 SNAP

-S <memory address> , <address argument> [, ...]

Each time the memory location specified by the memory address is encountered, the contents of the ranges specified by the address argument are typed at the terminal in hexadecimal form.

The SNAP can be used successfully only when the instructions at the SNAP location and at the next location always are executed consecutively. Thus, the instruction at a SNAP location cannot be either a skip or a transfer such that the instruction at the following location would not be executed consecutively. Execution of the SNAP command does not remove it from the user's environment. It is in effect each time that instruction at the specified location is executed. The user may type any character during the SNAP output to terminate the output and the DEBUG prompt is issued (as if a HALT occurred at the SNAP location).

NOTE

> DEBUG allows the user to set up to seven HALTs and/or SNAPs. If the user attempts to specify an eighth HALT or SNAP, DEBUG responds with an OV and refuses to accept the command.

For example, if the user wishes to see the contents of arithmetic register AC0 and the contents of memory addresses 145 through 148, each time the user routine reaches address 224A, the following SNAP command may be used:

```
-S224A, R0, 145:148 (CR)
```

The following is an example of the output that is printed by DEBUG each time the specified address is encountered:

```
CL    224A

R0000  1234
0145   FFFF  1234  5678  9ABC
-
```

typical data

1.5.7 GO

-G <memory address>

The GO command initiates transfer of control to the user's program at the location specified by the optional memory address. If no memory address is specified with the GO statement, the program continues at the last HALT location. All registers are restored in either case.

To start a user's program at location 2600, the following GO statement can be used:

-G2600 (CR)

To continue execution after a HALT, the following GO statement can be used:

-G (CR)

1.5.8 MOVE

-M <value> , <address argument>

A selected hexadecimal number or ASCII character pair is placed in the specified range of memory addresses or registers.

For example:

-M'XX', 250:25F (CR) loads all locations with XX

and -M 0, 250:25F (CR) puts zeros into the range of locations

1.5.9 NOTE

-N <comment>

The NOTE command permits the user to comment his debugging. All text prior to the carriage return or the line feed is printed on the terminal. No other action is performed.

For example, a NOTE comment may appear as follows:

-N A COMMENT MAY BE PLACED AFTER THE N (CR)
-N AND SUBSEQUENT LINES MUST ALSO BE (CR)
-N PRECEDED BY AN N. (CR)

1.5.10 FIND

-F <value> , <address argument>

The locations in the range of address argument are searched and the first location that is equal to the value is typed. If there is no match, DEBUG simply reprompts. The current location is either the location found or the last location searched.

To find the pair of ASCII characters 'LK' in the first 4K of memory, the following command may be used:

-F'LK', 0:FFF (CR)

DEBUG responds with the following output:

CL 0AAA if the characters are located at memory location AAA.

-

To find a hexadecimal number, the following command may be used:

-F 48, 0:FFF (CR)

The first 4K of memory is searched. If the number is not found, DEBUG reprompts.

1.5.11 RESET

-R

The RESET command causes all of the SNAPs or HALTs to be removed and the original code replaced.

For example:

-R (CR)

1.5.12 CARRIAGE RETURN (CR)

Typing only the carriage return causes typing of the current location (CL) in the format (ASCII/hexadecimal) of the last command.

1.5.13 LINE FEED (LF)

Typing only the line feed causes typing of the current location plus one (CL+1) in the current format. The current location is also increased by one.

1.5.14 BACK ARROW (←)

Typing only the back arrow causes typing of the current location minus one (CL-1) in the current format. The current location is decreased by one.

1.5.15 Summary of Commands

Table 1-1 lists the commands in alphabetical order.

Table 1-1. Summary of DEBUG Commands

Symbol	Command	Structure of Statement
A	ALTER	A { <memory address> <register address> } , <hexadecimal number> [, <...> <...>]
C	CHAR	C <address argument> [, <address argument> ...]
F	FIND	F <value> , <address argument>
G	GO	G <memory address>
H	HALT	H <memory address> , <hexadecimal number>
M	MOVE	M <value> , <address argument>
N	NOTE	N <comment>
R	RESET	R
S	SNAP	S <memory address> , <address argument> [, ...]
T	TYPE	T <address argument> [, <address argument> ...]
CR	Type Current Location (CL)	
LF	Type Next Location (CL+1)	
←	Type Previous Location (CL-1)	

Chapter 2

IMP-16 LOADERS

2.1 INTRODUCTION

The IMP-16 loaders are a compilation of programs that read and load one or more Relocatable Load Modules (RLMs), produced by the IMP-16 Assembler, into the main memory for execution. Each of these programs is introduced briefly below and then is described in subsequent sections of this manual.

Absolute Paper Tape Loader (ABSPT) is a stand-alone program that reads GENLDR (or any other single RLM not requiring relocation) from the paper tape reader and loads it into main memory for execution. ABSPT is permanently resident in the IMP-16 ROM. Paper tapes read by the IMP-16 are assumed to contain eight channels of binary data. An RLM appears as a series of records of binary data in standard format (IMP-16 Programming and Assembler Manual); each record is preceded by a Start of Text (STX) character and separated by Null characters.

ABSCR is a stand-alone program that reads any RLM not requiring relocation from the card reader and loads the program into the main memory for execution. The primary use of ABSCR is to load GENLDR. ABSCR is loaded into the main memory of the IMP-16L for execution by the Card Reader Bootstrap Loader (CRBOOT) but is permanently resident in IMP-16P ROM.

CRBOOT is a stand-alone program for the IMP-16L and is not required for the IMP-16P. It is bootstrapped directly into main memory for execution under control of IMP-16L equipment.

General Loader (GENLDR) is a self-contained IMP-16 program and performs the following functions:

- Reads and loads one or more RLMs from the card reader and/or the Teletype.
- Relocates the modules.
- Resolves external linkages between the modules.
- Provides descriptive information describing memory, globals.

GENLDR is command-driven and provides comprehensive control over the loading process. GENLDR has the capability to change the loading input device. Otherwise loading proceeds in sequence from the device upon which it was initiated.

2.2 ABSPT (ABSOLUTE PAPER TAPE LOADER)

ABSPT loads GENLDR or any other RLM (not requiring relocation) into the IMP-16 main memory from the paper tape reader.

2.2.1 Usage

The RLM tape loaded by ABSPT is an 8-channel tape, composed of successive RLM records, each preceded by a Start-of-text (STX) character. Since each record contains its own length, no extra characters may appear within records, but any character may appear between records.

2.2.2 IMP-16 Loading

ABSPT is resident in Read-only Memory (ROM) on the IMP-16. The procedure for loading paper tape is as follows:

1. Press the INITIALIZE button on the IMP-16 panel.
2. Place the RLM paper tape into the paper tape reader.
3. Press the LOAD PROG button on the IMP-16 panel.
4. Turn on the paper tape reader.

The RLM is loaded, the entry point address transferred to AC2, and the processing halted. At this point, the user can perform one of the following actions:

1. Press RUN to cause execution of the program loaded.
2. Alter the entry point address contained in AC2 and press RUN to cause execution to start at the modified entry point.
3. Load another RLM in the same manner as before. No resolution of inter-RLM linkages is performed; the user is cautioned to ensure that an RLM does not overlay a previously loaded module.

ABSPT checks only for a checksum error and halts if one is discovered. To retry the load, position the paper tape at the beginning of the record in error; press RUN; and turn on the reader. In order to ignore the error, press RUN.

2.3 ABSCR (ABSOLUTE CARD READER LOADER)

ABSCR loads one or more RLMs (which do not require relocation) into the main memory from the card reader. In the IMP-16L, CRBOOT is used to load ABSCR into memory; because of this close relationship, CRBOOT is described under the current heading. Refer to figure 2-1 for a description of a sample deck using CRBOOT and ABSCR.

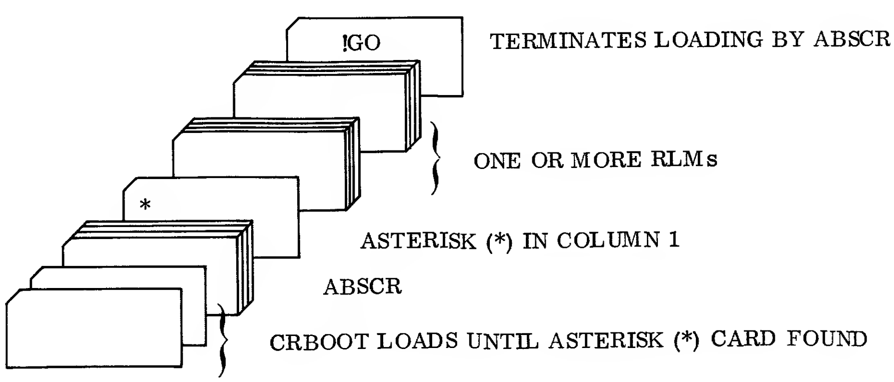


Figure 2-1. Example of Card Load by ABSCR

2.3.1 Usage

In the IMP-16P, ABSCR is resident in ROM. In the IMP-16L, it is resident in memory, starting at the location specified by the user, and using $E9_{16}$ words. ABSCR cannot load RLMs into memory location which it occupies.

The RLM card deck loaded by ABSCR is punched one card per RLM record; columns 73 through 80 of the card are ignored. Only the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and blank are allowed in columns 1 through 72; a blank is treated as a zero. Each record is the hexadecimal character equivalent of an RLM record as output by the IMP-16 Assembler (see IMP-16 Programming and Assembler Manual).

2.3.2 RLM Loading

The sequence of events for loading an RLM, which contains nonrelocatable coding, from the IMP-16L card reader is as follows:

1. Place CRBOOT in the card reader, followed by ABSCR, followed by the RLMs to be loaded, and a !GO card.
2. Press INITIALIZE.
3. If ABSCR is to be loaded at other than location 120_{16}, perform the following steps:
    a. Set Mode Switch to ACO.
    b. Set ABSCR load address into switches.
    c. Press LOAD DATA.
4. Press AUX1.
5. Press RUN.

The sequence of events for loading RLMs from the IMP-16P card reader is as follows:

1. Place the RLMs into the card reader followed by a !GO card and ready the card reader.
2. Press INITIALIZE.
3. Set MODE Switch to PC.
4. Set X'7F00 into switches.
5. Press LOAD DATA.

NOTE

The following step must be performed; otherwise the system halts.

6. Set MODE Switch to PROG DATA.
7. Press RUN.

As shown in figure 2-1, ABSCR continues to load RLMs until the !GO card is encountered. When this occurs, if a nonzero entry point is specified in the last RLM, ABSCR loads AC3 with a 1 to indicate that the load device is the card reader and transfers control to the specified entry point.

If the last specified entry point is a '0' (supplied by the assembler as a default value), ABSCR halts. (See Error Code 5, below.) At this point, the user may enter the correct entry point into AC1 and press RUN.

When the user loads more than one RLM, no resolution of inter-RLM linkages is performed; the user is cautioned to ensure that an RLM does not overlay a previously loaded module.

When ABSCR detects an error, it places an error code in AC0 and halts execution. The following codes comprise the error codes:

Code	Description
1	I/O Error - A transmission error or data overrun condition occurred on the card reader. In the IMP-16P, this code is the result of a motion error. The status word returned from the reader is placed in AC1 before halting. To reread the card, replace the card in the reader and press RUN.
2	Invalid Character - Only punches for 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and blank are allowed in card columns 1 through 72. Correct the card; replace it in the card reader; and press RUN.
3	Checksum - The checksum calculated by ABSCR did not match that found on the last card read, indicating either a read error or a bad card. Correct the card; replace it in the reader; and press RUN.
5	Invalid Entry - The last END record read contained an entry-point address of '0' and a !GO card was read. Place the correct entry-point address in AC1 and press RUN.

2.4 CRBOOT (CARD READER BOOTSTRAP LOADER)

CRBOOT is the card reader bootstrap program for the IMP-16L. Its sole purpose is to load a single program that is punched onto cards in a specific format. Typically, it is used to load ABSCR.

2.4.1 Usage

CRBOOT reads successive cards containing only hexadecimal characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, or blank in columns 1 through 72; blank is treated as a zero. Characters are converted to their 4-bit hexadecimal equivalents, packed 4 characters to a word, and stored in successive memory locations. CRBOOT passes control to the loaded program (at the first location loaded) when it reads a card with an asterisk (*) in column 1. (The rest of that card is ignored.) When it transfers control, interrupts are disabled.

2.4.2 Bootstrap Procedure

The following procedure should be performed to bootstrap from the card reader:

1. Press INITIALIZE.
2. Place CRBOOT in the card reader, followed by deck to be loaded (typically ABSCR) and a card containing an asterisk (*) in column 1.
3. If deck is loaded at other than location 120_{16}, perform the following steps:
    a. Set Mode Switch to AC0.

b. Set load address into switches.

c. Press LOAD DATA.

4. Press AUX 1.

5. Press RUN.

Note that because CRBOOT resides in memory words 0 through $4F_{16}$ and uses words 50_{16} through $9F_{16}$ as an input buffer, CRBOOT cannot load a program into locations 0 through $9F_{16}$.

CRBOOT halts at location 0010_{16} if it detects a transmission error or a data overrun condition on the card reader; the entire load process then has to be repeated.

CRBOOT halts at location 0020_{16} if it detects an invalid punch. (Only punches 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and blank are allowed.) Correct the card and repeat the entire load process.

2.4.3 Format of CRBOOT

Since CRBOOT is read by the IMP-16L equipment card reader bootstrap, it must be punched in a special format.

Each (odd, even) pair of card columns is packed into one memory word. Rows 2 through 9 of the odd-numbered column are moved into bits 15 through 8; rows 2 through 9 of the even-numbered column are moved into bits 7 through 0. The hardware bootstrap reads one card, packs it into words 0 through 39_{10} (27_{16}), and passes control to word 0.

CRBOOT is a 2-card bootstrap (see figure 2-2). The first card contains the instructions necessary to read the second card, which has the same format as the first card.

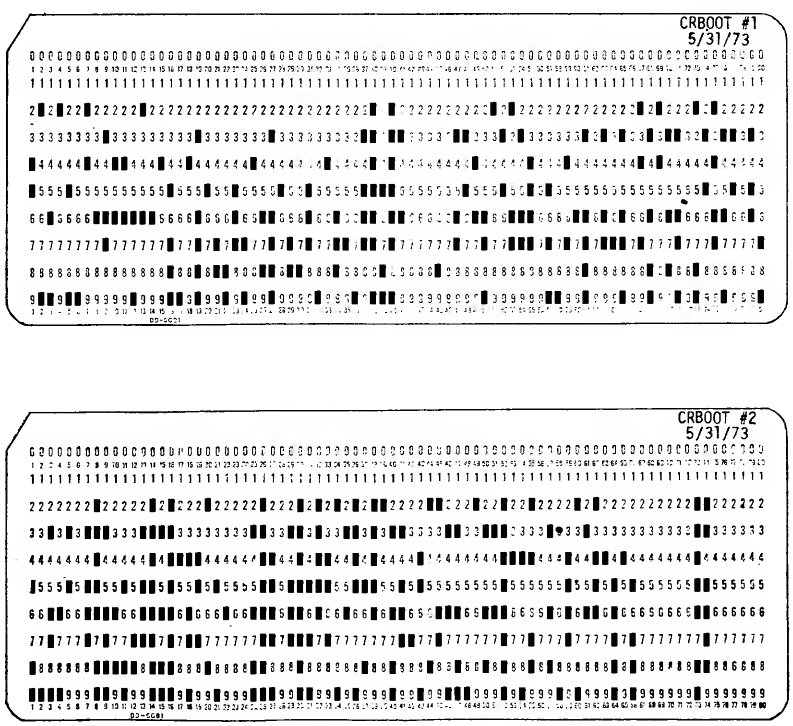


Figure 2-2. Program Cards for CR Bootstrap

2.5 GENLDR (GENERAL LOADER)

GENLDR is a stand-alone IMP-16 program that loads one or more Relocatable Load Modules (RLMs) produced by the IMP-16 Assembler, performs relocation and resolves external linkages, and loads the RLMs into main memory for execution.

Each RLM must be transcribed to punched cards or paper tape (ordinarily, on the same machine used for assembling IMP-16 Assembly Source programs). These RLMs, as well as the commands that control the loading process, are then input to GENLDR. RLM record formats are described in appendix A of the IMP-16 Assembler Manual.

The paragraphs that follow describe the commands to control GENLDR and the input sequences required to load an executable program into the IMP-16 main memory. Error messages and diagnostic output of GENLDR are also described.

2.5.1 Usage

GENLDR is a nonrelocatable stand-alone IMP-16 program that must be loaded into memory by one of two absolute loaders: (1) ABSCR allows GENLDR to be input from cards and (2) ABSPT, from paper tape. Once loaded, GENLDR can accept input from either cards or paper tape; although, initially, it accepts input from the device from which it was loaded.

GENLDR occupies approximately 1700_{10} words of memory and is typically loaded into upper memory. Programs cannot be loaded by GENLDR into memory that it occupies or uses for the symbol table it generates. However, GENLDR allows the user full use of base page. The memory layout is described in figure 2-3.

The IMP-16 Assembler allows the user to allocate portions of his program in three ways:

- At an absolute memory location
- Relative to the origin of the base sector
- Relative to the origin of the top sector

Typically, absolute allocation is employed to assign locations dependent upon equipment (for example, interrupt entrance address) or to communicate with special-purpose routines. The base sector must be located such that it is contained within the first 256_{10} (100_{16}) locations of memory and typically contains data and pointers necessary for inter-RLM communication. The top sector may reside anywhere in memory (subject to the limitations mentioned above) and normally contains the main portion of the RLM. Care must be exercised to ensure that an absolute sector does not overlay a previously loaded base sector or top sector. (See !OBS and !OTS commands in the following paragraphs.)

Two other limitations are imposed upon the base sector by the IMP-16 computer architecture and the method for resolving certain external linkages. First, any base sector variable that is referenced by an indexed instruction must be allocated to one of the first $7F_{16}$ locations of memory. Second, in resolving certain external linkages, GENLDR may force an indirect reference to a global variable through a pointer in the memory area FF_{16} and downward.

The area of IMP-16L memory between locations 100_{16} and $11F_{16}$ is used by the control panel service routine and may not be used by the user. Above address FF_{16}, loading is limited, only within the area occupied by GENLDR and the symbol table it generates. (This area may be used by the loaded program, after it receives control from GENLDR.)

As an entry point, GENLDR selects the last nonzero value specified for the set of RLMs loaded. The entry point for any particular RLM, if specified, appears in the END record of that RLM. If the user desires, he may override the entry point selected by GENLDR by specifying the desired entry point in the !GO command (paragraph 2.5.17). If neither of these methods is chosen, GENLDR prints an "ENT" error message and prompts for a new command.

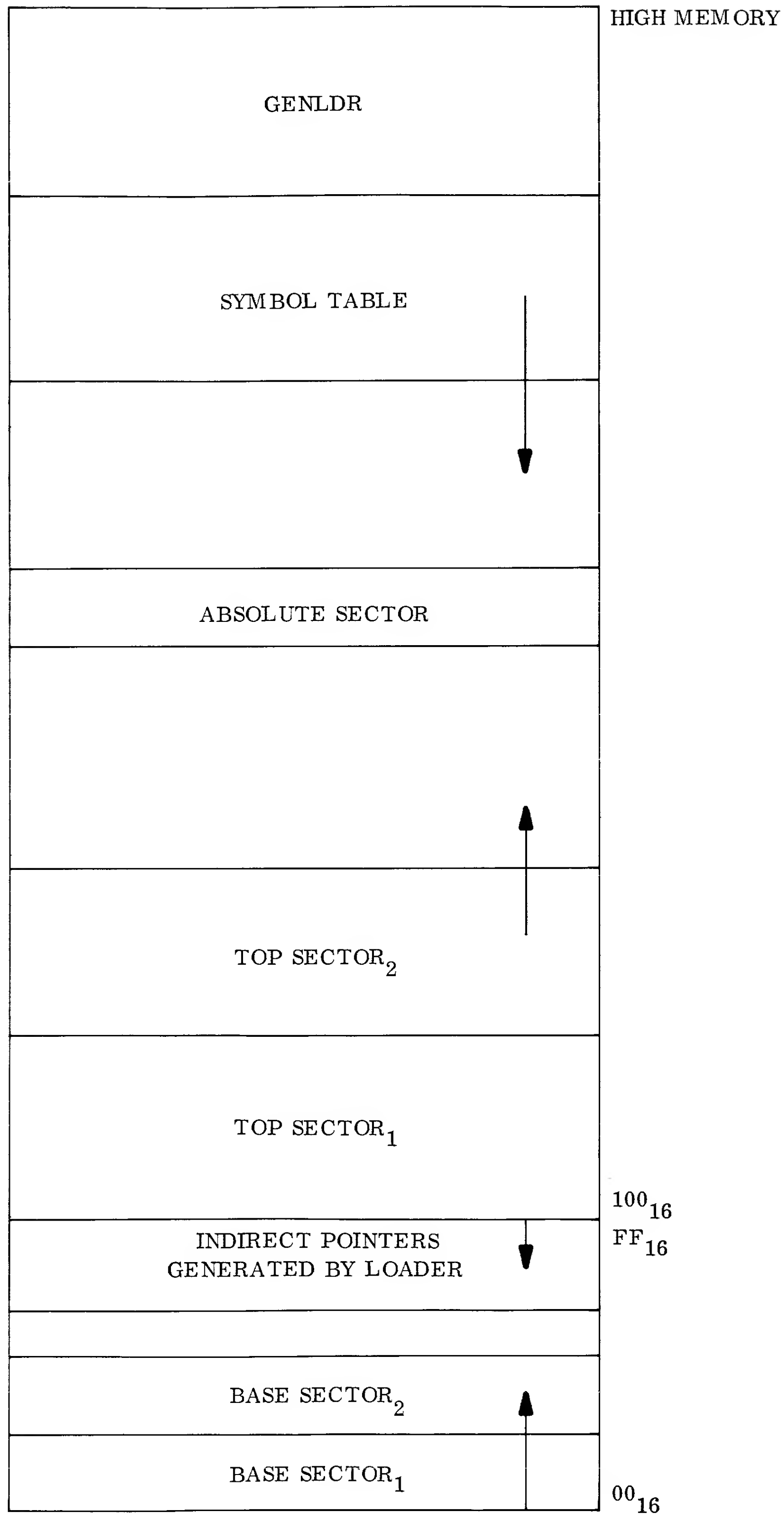


Figure 2-3. Memory Map

2.5.2 GENLDR Input

GENLDR is command-driven. It reads commands and RLMs from either cards or paper tape, and commands are available to switch between input devices. (See !CR and !TTY, paragraphs 2.5.9 and 2.5.10). GENLDR does not recognize any distinction between the Teletype paper tape reader and the Teletype keyboard; therefore, the user may type in his commands at the keyboard and input the RLM from paper tape. Commands entered either on paper tape or the keyboard are echoed back to the Teletype printer; the RLM itself is not echoed.

Commands entered on punched cards must contain an exclamation point (!) in column 1. When the command is entered from the Teletype, GENLDR types the exclamation point to prompt for a command. The user should not type the exclamation point.

Input lines on the Teletype can be terminated by either a carriage return or a line feed. GENLDR supplies a line feed if a carriage return is issued, or a carriage return if a line feed is issued. GENLDR recognizes the following special characters when reading from the Teletype:

←	Backspace character
ALT MODE	Delete entire line
RUB	Rubout character (ignored)
	Null character (ignored)

A maximum of 72 characters is allowed in one Teletype input record; excess characters are ignored.

2.5.3 GENLDR Output

GENLDR may be directed to print information descriptive of the loading process. The title information, base sector limits, top sector limits, absolute sector limits, indirect pointer limits, and entry point address of each RLM may be printed on the Teletype (see !LM command). The user may also request the printing of the symbol table, or only those entries of the symbol table that refer to undefined (U) or multiply-defined (M) symbols (see !SY and !ER commands in the following paragraphs).

If !LM is issued, GENLDR types the following information at the end of each RLM:

```
MNEMONIC                        QUALIFYING STRING   AAAA   BBBB

BS=XXXX:XXXX  TS=XXXX:XXXX  AS=XXXX:XXXX  PTR=XXXX:XXXX  ENT=XXXX
```

where:

- MNEMONIC is the name of the RLM from the TITLE record.
- QUALIFYING STRING is the qualifying string from the TITLE record.
- AS specifies the low and high addresses of the absolute sector.
- BS specifies the low and high addresses of the base sector.
- TS specifies the low and high addresses of the top sector.
- AAAA and BBBB are the RLM source and object checksums, respectively.
- PTR is the number of indirect pointers generated.
- ENT is the entry address from the END record.

All numbers are printed in hexadecimal notation. If !NLM is the last command issued (of !LM or !NLM), when !GO is executed, one line of the above format is output containing composite limits of all RLMs over the scope of the !NLM command.

If !SY or !ER is specified (or defaulted), GENLDR prints symbols as follows:

SYMBOL XXXX F

where:

- SYMBOL is the symbol name.
- XXXX is the hexadecimal address of the symbol.
- F is one of the following:

  M - multiple-defined symbol

  U - undefined symbol

  blank - defined symbol

2.5.4 GENLDR Commands

All commands must begin in column 1 of the input record. One or more blanks must separate the command from an operand. Unless otherwise specified, where the term <hex-value> is used below, it represents a hexadecimal number in the range 0000 to FFFF. Leading zeros need not be specified, but no more than four hexadecimal characters can be given for <hex-value>.

2.5.5 !OBS - Origin Base Sector

!OBS <hex-value>

NOTE

> 0_{16} <= hex-value <= FF_{16}. If this command is not given, the first base sector is loaded at location 10_{16}.

The origin for the next base sector is set to <hex-value>. If this command is not specified, the next base sector is loaded immediately following the previous base sector. This command should be used to prevent loading a base sector on top of an absolute sector.

2.5.6 !OTS - Origin Top Sector

!OTS <hex-value>

NOTE

> The highest value of <hex-value> is a function of the memory available, and must not cause overlaying of the locations occupied by the symbol table or GENLDR. If this command is not given, the first top sector is loaded at location 120_{16}.

The origin for the next top sector is set to <hex-value>. If this command is not specified, the next top sector is loaded immediately following the previous top sector. This command should be used to prevent loading a top sector on top of an absolute sector.

2.5.7 !RLM - Relocatable Load Module Identifier

!RLM

This command must precede each RLM to be loaded. The RLM is loaded from the same device from which the RLM command is entered.

2.5.8 !CLR - Clear Memory

!CLR

Memory below GENLDR is cleared to zeros if this command is issued before the first RLM is loaded. After loading is completed (that is, a !GO command is read), memory containing the symbol table and GENLDR is zeroed; this latter function is performed even if the !CLR command is issued after an RLM is loaded.

2.5.9 !CR - Read Input from the Card Reader

!CR

Subsequent input is accepted from the card reader.

2.5.10 !TTY - Read Input from the Teletype

!TTY

Subsequent input is accepted from the Teletype. Teletype input is accepted from either the paper tape or the keyboard, but only commands are echoed to the Teletype printer.

2.5.11 !SY - Print the Symbol Table

!SY

The symbol table is printed upon execution of this command.

2.5.12 !ER - Print Symbols in Error

!ER

Multiply-defined and undefined symbols are printed when this command is read.

2.5.13 !LM - Print Limits

!LM

As each RLM following the !LM command is loaded, GENLDR prints the name, checksums, base sector limits, top sector limits, and absolute sector limits of the RLM. Each time RLM limits are printed, they are reset so that a record of areas occupied by only single RLMs is maintained by the program. The default command is !NLM.

2.5.14 !NLM - Don't Print Limits

!NLM

The name, base sector limits, and top sector limits are not printed. Instead, when a !GO command (2.3.4.13) is executed, GENLDR prints the combined base sector limits, top sector limits, and absolute sector limits of all programs loaded. The default command is !NLM.

2.5.15 !SQ - Check Sequence Numbers on Input Deck

!SQ

After this command is executed, the sequence number field of each input card (columns 73 through 80) is tested to ensure that the sequence numbers contained therein appear in ascending order from one card to the next. If they do not appear this way, an error message is printed. The default command is !NSQ.

2.5.16 !NSQ - Do Not Make Sequence Number Check

!NSQ

Execution of this command nullifies the execution of an !SQ command. The default command is !NSQ.

2.5.17 !GO - Execute the Loaded Program

!GO ⟨hex-value⟩

The entry point specified in the last RLM loaded can be overrriden by specifying the entry point address (" ⟨hex-value⟩ "). If ⟨hex-value⟩ is omitted, the last nonzero entry point specified is executed. If no nonzero entry point is specified and no value appears in the command, GENLDR prints an error message and returns to the command mode. If a !CLR command is previously read, the symbol table and memory containing GENLDR are zeroed before execution of the loaded program. Before transfer to the entry point, the combined limits of all programs loaded are printed on the Teletype if the individual program limits are not printed.

2.5.18 Messages

The following messages may be output by GENLDR:

GENERAL LOADER (REV, X) READY.

GENLDR is ready to accept commands. X is the current revision level of GENLDR.

CMND The command is invalid or unrecognized. Reenter the command.

CHAR An RLM record contains characters other than 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Correct the record; reload; and press RUN.

SEQ Record sequence error. The correct sequence is (1) Title Record, (2) Zero or more Symbol Records, (3) Zero or more Data Records, and (4) End Record. Correct record sequence; reload; and press RUN.

CKSM Checksum error on the next-to-last record read. If the checksum field is 0000, no checksum test is made. The read may be retried. Reload the record and press RUN.

NMBR A sequence number error is detected. Place the input cards in the correct order and restart, or continue.

BSOV Base sector overflow. The run must be restarted, but the error may be corrected by proper use of OBS and OTS commands.

TSOV Top sector overflow. The run must be restarted, but the error may be corrected by proper use of OBS and OTS commands.

SYMB Symbol table overflow. Too many external symbols defined. The run must be restarted, but the error may be corrected by proper use of the OTS command.

ADDR Addressing error. This error occurs under the following conditions and the run must be restarted:

1. Attempting to reference an indirect pointer generated by the assembler which, because of relocation, is forced to an address greater than 255_{10} (FF_{16}).

2. Using an index register in an instruction referencing a base sector variable allocated to a memory address between 128_{10} (80_{16}) and 255_{10} (FF_{16}).

3. Attempting to use an index register in an instruction referencing an undefined external variable.

4. Referencing an undefined external variable in an instruction which either is flagged indirect already or cannot be so flagged.

EXTN Unable to locate external symbol in Symbol Table. This error may be caused by attempting to load an RLM with some missing symbol records or by an erroneous patch which looks as if it is referencing an illegal external reference number. The run must be restarted.

AREA Loading in illegal area (possibly on top of the loader). Restart with valid !OBS or !OTS commands.

MEM Memory size exceeded. Loading into nonexistent memory. Recovery not possible, but error may be corrected by proper use of OBS and OTS commands.

SYST System error caused by a malfunction in system software or hardware. Recovery not possible.

PNCH Invalid punch in input record. Only the characters: blank, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, ..., X, Y, and Z accepted. Correct; reload record; and press RUN.

CRDR 1. Card reader is offline. Place reader online and press RUN.

2. Transmission error or data overrun on card reader. The status word returned by the card reader is in AC0. Replace card in reader and press RUN.

ENT No entry point specified for program. GENLDR transfers control to the Teletype for new command.

DROP Card dropped out of deck (that is, sequence number incremented by more than 10 and !SQ in effect). Check last two cards read. If there are no cards missing between them, press RUN; otherwise, correct card deck and reload.

PTCH nnnnnnnn

A patch card (card with sequence number that is not a multiple of 10) with sequence number 'nnnnnnnn' was processed.

2.5.19 Sample GENLDR Run

The sequence of a sample IMP-16L GENLDR run is shown in figure 2-4. The sequence starts with CRBOOT and proceeds through the steps shown to !GO, upon which execution of GENLDR starts at location 288_{10} (120_{16}) When GENLDR is run on an IMP-16P, CRBOOT and ABSCR are not required in the sample deck.

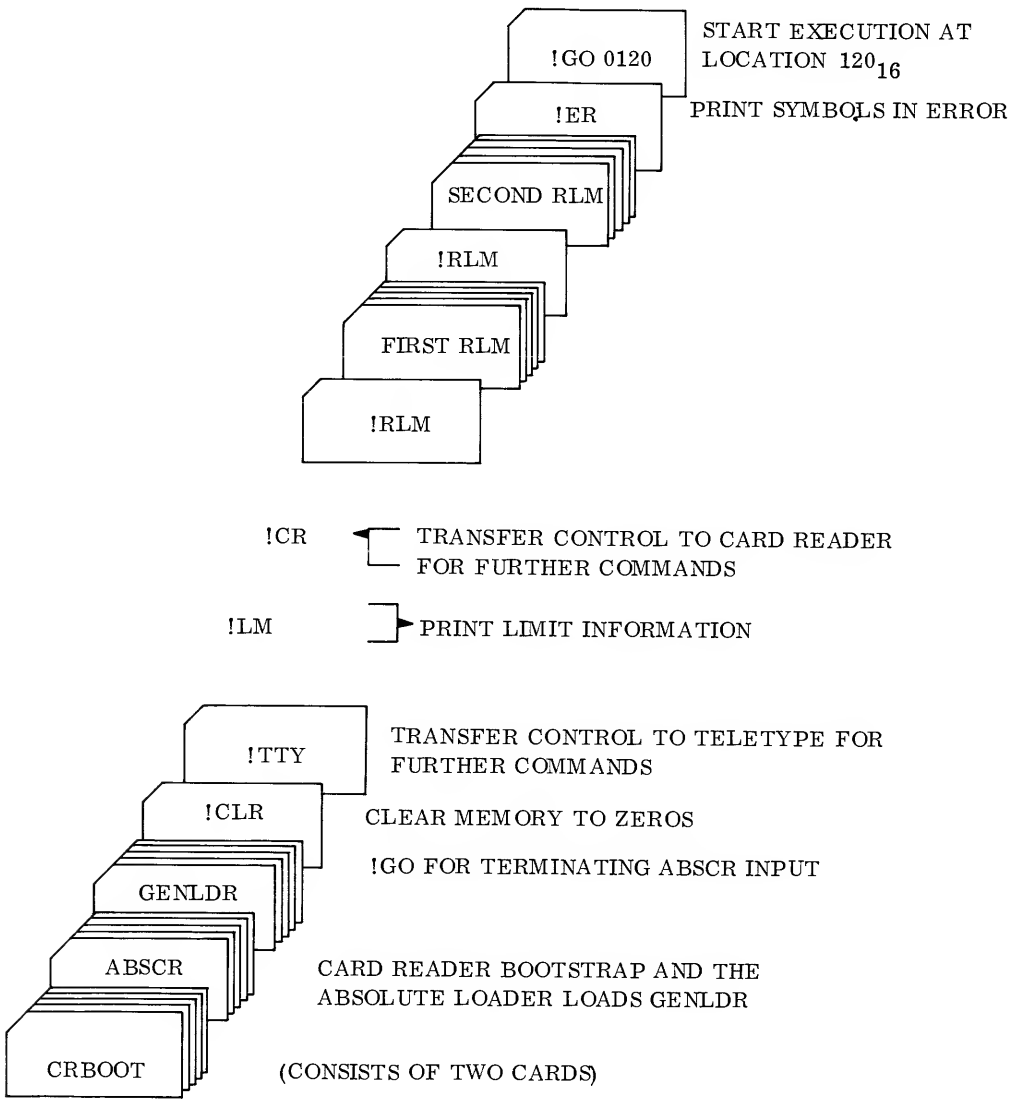


Figure 2-4. Sequence of Sample IMP-16L GENLDR Run

Chapter 3

TELETYPE I/O ROUTINES

3.1 INTRODUCTION

STTYIO contains five Teletype input/output routines assembled in one relocatable object program. Each routine performs one commonly used Teletype function.

3.2 USAGE

A transfer vector is reserved in base page to establish a fixed interface between user and STTYIO

3.2.1 Functions

The five Teletype functions in STTYI 0 are:

- SETPL — Resets the Teletype and initializes the other four routines for IMP-16L/16P operation.
- INTEST — Tests for Teletype input.
- PUTC — Transmits bits 0 - 7 of Accumulator 0 (AC0) to the Teletype.
- GETC — Transfer from Teletype to bits 0 - 7 of Accumulator 0 (AC0).
- GECO — Same as GETC plus an echo of the character on the Teletype printer.

3.2.2 Communications

The user may call these routines by using the JSR@ instruction and the transfer vector reserved in base page. Addresses in base page are reserved as follows:

```
000B    SETPL
000C    INTEST
000D    PUTC
000E    GETC
000F    GECO
```

For example, to transmit bits 0 - 7 of AC0 to the Teletype, the following instruction may be included in the instruction stream:

```
JSR     @X'D
```

When using INTEST, return from the routine is as follows:

```
CALL+1  INPUT AVAILABLE FROM TELETYPE
CALL+2  NO INPUT FROM TELETYPE
```

A typical user's program using INTEST may appear as follows:

```
JSR     @X'C
JMP     ATTINPUT     ;ATTEMPT TO INPUT
JMP     NOINPUT      ;NO INPUT FROM TELETYPE
```

Return from all the other routines is RTS 0.

3.2.3 Limitations

When STTYI0 is loaded in memory, locations X'B through X'F in base page are reserved locations. SETPL must be called prior to use of any of the other four routines. Registers and flags are not altered by any of the five routines except AC0 in GECO and GETC. The stack is pushed three levels deep during execution of these routines.

These routines are dependent on system speed. They are good only at normal system speed (1.4 μsecond per microcycle).

3.3 LOADING

STTYI0 may be loaded by the absolute card reader loader, absolute paper tape loader, or the general loader. Refer to chapter 2 of this manual for additional information concerning IMP-16 loaders.

3.4 STORAGE REQUIREMENTS

Base Page	X'B through X'F
Top Sector	STTYI0 is currently assembled top sector relocatable and occupies 123_{10} ($7B_{16}$) words.

Chapter 4

FIRMWARE PAPER TAPE GENERATION

4.1 INTRODUCTION

PROMP is a program that generates paper tapes for PROM programming. It has the following capabilities:

- Punch PN format PROM tape.
- Punch BC (binary complemented) format PROM tape.
- Punch paper tape RLM from card RLM.

PROMP output tapes for ROM programming are used only in ROMs programmed 8-by-256 bits (used in multiples of two).

4.2 PROGRAM ENVIRONMENT

4.2.1 Program Loading

PROMP may be loaded by GENLDR or by the absolute loader (ABSCR), when PROMP is in card deck object format, and by the LOAD PROG function in the IMP-16 when PROMP is in paper tape object format. PROMP uses the Teletype I/O functions provided in STTYI0 and thus requires that STTYI0 also be loaded into memory.

4.2.2 Memory Requirements

PROMP requires 655_{16} words of top sector memory.

4.2.3 Program Messages

To guide the user in the use of PROMP, query messages are typed. There are four general messages corresponding to the four possible program states. Initially, PROMP types:

```
NSC IMP-16 FIRMWARE PAPER TAPE GENERATOR
OUTPUT TYPE:
```

The user must then respond with one of the following codes:

PN - For PN format PROM tape.
BC - For binary (complemented) format PROM tape.
OM - For card RLM to tape RLM conversion.

All responses must be terminated by a carriage return. For the OM option, PROMP types the following message and goes into an output state:

```
TURN PUNCH ON
MAKE CARD READER READY
```

At this point, RLM records are converted from card to paper tape. After processing the END record, PROMP goes into a wait state. The user may now turn off the Teletype punch. Typing any key returns the PROMP program to the initial state.

In either PN or BC options, PROMP goes into an input state. The following message is typed:

INPUT DEVICE:

The user may respond with one of the following codes:

CR - Card Reader
PT - Teletype
ME - Memory

ME Option. If this device is selected, PROMP then types:

SPECIFY MEMORY --

The user must then type the memory range, where the user program is located. A range is designated by the start address and the last address, delimited by a colon (:); the range must be in blocks of 256_{10} or 512_{10} words.

Example:

SPECIFY MEMORY --FF00:FFFF

CR Option. PROMP types the following message:

MAKE CARD READER READY
TO LOAD RLM

The card reader must be turned on at this point.

PT Option. If this device is selected, PROMP types the message:

MAKE TAPE READER READY
TO LOAD OM

After the loading process, when the END record is recognized, PROMP types:

TURN READER OFF

The user must turn off the reader and press any key to initiate the output process.

After the loading process, PROMP goes into an output state and types:

OUTPUT OPTION:

The user must respond with one of the following codes:

CR	Default, options 1 through 4, below.
1	First 256 words, left byte.
2	First 256 words, right byte.
3	Second 256 words, left byte.
4	Second 256 words, right byte.

If more than one option is desired, the user may type options consecutively, delimiting with either a comma (,) or space. After the option is typed, PROMP comes back with the message:

TURN PUNCH ON
HIT ANY KEY TO START

After the output process, PROMP goes into its last state, the wait state. This wait state is provided so that the user can turn off the punch before any message is typed by PROMP. To return to the initial state, the user must press any key on the keyboard.

Typing CTRL/D when PROMP is waiting for a response transfers the user to the initial state. Leaders/trailers are punched automatically. If PN is chosen as an output type code, pressing any key during the output process causes an interrupt of the current output option and initiates processing of the next output option.

4.3 INPUT/OUTPUT FORMATS

4.3.1 Input Formats

PROMP accepts object programs in RLM format and memory contents for input. Input device may be a card reader, Teletype, or memory.

4.3.2 Output Formats

There are three possible output formats from PROMP: PN and BC for PROM tapes, and RLM tapes.

PN Format. In this format, 1 (one) bit is represented by one frame P (for 1) and N (for 0) in ASCII format. Each 8-bit block is preceded by the character B and followed by F. CR or LF precedes each B character.

NOTE

This format should be used when ordering ROMs or PROMs from National Semiconductor Corporation.

BC Format. In this format, 8 bits (may be left or right) is represented by one frame. The frame contents is the binary complement.

RLM Format. Refer to the IMP-16 Programming and Assembler Manual and chapter 2 of this manual for a complete description of RLM object programs.

4.4 USAGE

By typing the right codes in response to PROMP queries, the user can use PROMP in a variety of ways. The following table shows all possible input/output options.

Table 4-1. Input/Output Options

Function	Input			Output		Output Option Codes
	Option Code	Input Device	Input Format	Option Code	Output Format	
RLM to PROM	CR	Card Reader	RLM	PN BC	PN Binary	CR, 1, 2, 3, 4
	PT	Teletype	RLM	PN BC	PN Binary	CR, 1, 2, 3, 4
	ME	Memory	Binary	PN BC	PN Binary	CR, 1, 2, 3, 4
Card RLM to Tape RLM		Card Reader	RLM	OM	RLM	Not applicable

Chapter 5

EDIT16

5.1 INTRODUCTION

EDIT16 is a paper tape source editor program, used with the IMP-16L/16P processors. EDIT16 enables editing of a previously prepared source program (or any text) or generating and editing new text. Once loaded, the program is self-starting and provides approximately 4000 characters of working storage in a 4K processor.

The normal editing procedure is to input text, edit the text, and output the edited text. Refer to paragraph 5.6 for a sample edit run.

Prepared text, in punched paper tape format, is read in through the Teletype paper tape reader. New text is generated by typing lines of text on the Teletype keyboard. A line of text is a string of characters followed by a CR character. Output text is punched on the Teletype paper tape punch.

EDIT16 commands are line oriented. Character editing capability is provided through the use of the "Modify Line/String" command. Automatic line renumbering is performed when lines are inserted, deleted, and moved.

Appendix E contains a table of the symbol meanings that are used in the discussions that follow.

5.2 COMMAND MODE AND TEXT MODE

The EDIT16 is in the command mode when it types a '?' prompt character and waits for a command input. Most EDIT16 commands operate in this mode, but there are a few commands that require additional information for the command to be carried out properly. In getting this additional information, EDIT16 goes into a text mode. Text mode is terminated by either a CTRL/Q or a CR depending on the command being processed. Command mode always follows successful command processing.

If an invalid EDIT16 command is typed, the message ERROR is typed, followed by CR/LF, and then the prompt character (?). Note that only one command per line is accepted.

All EDIT16 commands are terminated by a CR. If the command is not a text-type command, processing starts after the CR character is typed and EDIT16 remains in command mode. If the command is a text type command, typing CR causes EDIT16 to go into text mode.

Typing CR causes EDIT16 to respond with CR/LF; that is, a carriage return followed by a line feed. Note that when the keyboard is used as an input, CR signals the end of a line and also appears in the edit buffer as the last character of a line.

5.3 COMMAND DELIMITERS

5.3.1 Arguments

Arguments are used in EDIT16 commands to specify portions of the edit buffer upon which the command operates. When a command requires an argument but none is typed, default values are called upon and these vary according to the command. EDIT16 uses the following arguments:

n   Is an unsigned decimal number from 1 to 9999. It denotes existing line numbers in the edit buffer.

/ — Specifies the high and low values of a range argument.

TO — Must be followed by n, and operates on n such that n signifies the destination in a transfer type command.

m — Number of lines count: usually used in addition or insertion of new lines.

' — Single quote: used to enclose a string of characters for commands that require string arguments.

The first four symbols may be combined in different formats to help the user achieve specific tasks. Valid combinations are as follows:

m TO n — Usually used to insert new lines into existing text. m gives the number of lines to insert before line number n.

TO n — Same as above except that the number of lines to insert before line number n is not a fixed amount. CTRL/Q is used to terminate insertion.

n_1 TO n_2 — Used in copy and move line commands. n_1 is the line to be moved or copied and n_2 is the destination line number. Note that both n_1 and n_2 must be existing line numbers.

n_1/n_2 — Used to specify a line number range.

5.3.2 Command Properties

All commands are line oriented. Line numbers specified in a command argument must be lines existing in the edit buffer. Specifying non-existent lines causes abnormal EDIT16 execution, which may result in program crash.

Arguments must be separated from the command by at least one space; although, some commands may execute properly even without command and argument separation.

All valid commands are two characters long.

5.3.3 The Edit Buffer

The edit buffer contains the source text which is being edited. Each line is composed of a line number, line source, and a CR character. Source is packed, two ASCII characters per word, and repeated spaces are packed using a repeat count. The edit buffer is located on top of the EDIT16 code such that its size is dependent on the machine memory space.

5.4 COMMAND SET

Commands are grouped into three sets: Input/Output Commands, Text Modify Commands, and Search Commands.

5.4.1 KB - Keyboard Read

The KB command is used to enter text into the edit buffer from the Teletype keyboard. EDIT16 types -> whenever it is ready to accept a new line. Typing CTRL/Q as the first character in a new line terminates the keyboard entry and EDIT16 goes to command mode.

Example:

	? KB CR	KB without an operand appends all text entered from keyboard.
->	FIRST LINE CR	
->	CNTL/Q ***	
	? KB 1 CR	Append one line to edit buffer.
->	APPEND ONE LINE CR	
	? KB 2 TO 2 CR	Insert two lines before current line 2.
->	FIRST INSERT CR	
->	SECOND INSERT CR	
	? KB TO 3 CR	Insert all text entered before current line 3.
->	THIRD INSERT CR	
->	FOURTH INSERT CR	
->	FIFTH INSERT CR	
->	CNTL/Q ***	
	?	Next prompt.

A description of the symbols used in these examples is included in appendix E. Typing CNTL/Q when EDIT16 is waiting for a command initiates the keyboard input mode.

Example:

? CNTL/Q

->

The current line being entered is ignored if CNTL/Q is entered before the CR . A back arrow ← deletes the last character typed.

5.4.2 RT - Read Paper Tape

The RT command enters text into the edit buffer from the Teletype reader, but does not print the lines. This command processes input lines, similar to the KB command. To terminate text entry when no fixed number of lines is specified in the argument, the user must turn off the reader and enter a CNTL/Q on the keyboard. The same action may be used to abort the entry. If the reader is turned off in the middle of a line, only the last complete line is transferred to the buffer.

If the edit buffer becomes full when entering lines using the KB, RT, and RC commands, the message:

BUFFER FULL

is typed and EDIT16 goes into command mode. Again, an incomplete line is not entered to the buffer.

Example:

? RT 2 CR	Append two lines to the edit buffer. This message is typed after the second line is read.
TURN READER OFF NOW	

5.4.3 RC - Read Card

The RC command must be issued only when applicable. That is, on systems equipped with a card reader. It reads a card and treats it as one line. Arguments applicable to the KB command (refer to paragraph 5.4.1) apply here also.

5.4.4 LS, LF, LL - Teletype List

The LS command lists text on the Teletype. Listed lines are numbered. LF lists line number 1, the first line of text. LL lists the last line of text.

Example:

? LS 2/4, 6 (CR)	Lists lines 2 through 4, and line 6 on the Teletype.
? LS (CR)	Lists entire text.

To interrupt the list, press any key on the Teletype keyboard.

5.4.5 PT - Punch Paper Tape

The PT command is used to punch text on paper tape. Immediately following the PT command, the following message is typed:

TURN PUNCH ON

EDIT16 then goes into the wait state. To continue operation, turn on the punch and press any key on the keyboard. To interrupt the punch, press any key. The punch operation is interrupted immediately. At the end of the punch operation, EDIT16 goes to a wait state. Turn off the punch and press any key. Line numbers are suppressed.

Example:

? PT 1/3 (CR)	Punch lines 1 through 3.
? PT (CR)	Punch entire text.

5.4.6 TL - Punch Leader/Trailer

The TL command punches approximately 5 inches of null characters for use as trailer or leader. Note that the PT command does not provide for a leader or trailer.

5.4.7 HP - High Speed Printer List

The HP command must be issued only where applicable. That is, for systems equipped with a high-speed printer. HP provides the same output list format as the LS command (refer to paragraph 5.4.4).

5.4.8 MD - Modify Line

The MD command enables character editing in a line. Characters may be inserted or deleted and lines may be truncated or extended.

Examples:

```
        ? MD 2 (CR)                   Modify line 2.
      2   SECOND LINE                 Line 2 is typed back.
ALTERS ?  SECOND TEST LINE (CR)       Line 2 is extended.
          SECOND TEST LINE            Line 2 is typed back.
ALTERS ?  (CR)                        (CR) first character typed, means end of modify
        ?                             line command.
```

(CTRL/Z) followed by any character 'C' advances the carriage to the first occurrence of 'C' in that line.

Example:

```
        ? MD 2 (CR)                   Modify line 2.
      2   SECOND TEST LINE            Line 2 is typed back.
ALTERS ?  (CTRL/Z) D                  (CTRL/Z) , then D advances the carriage to the
                                      first occurrence of the character D. (CTRL/Z)
                                      and D are not echoed but the line is.

ALTERS ?  SECON
               ^
               |_____________________ Carriage stops here, where the D would have been.
```

(CTRL/Z) 'C' may be issued at any point within the line as long as 'C' is present in the columns between the carriage and the last character of the line.

Typing (CTRL/X) on any column deletes the character in that column. A '↑' is echoed on the printer for each (CTRL/X).

Example:

```
      2   SECOND TEST LINE            'TEST' is to be deleted.
ALTERS ?  SECOND ↑↑↑↑↑ (CR)
      2   SECOND LINE
```

Typing (CTRL/A) inserts all characters typed after it, up to, but not including a (CR) . Characters inserted are enclosed by '<' and '>' .

Example:

```
        ? MD 2 (CR)                   Modify line 2.
      2   SECOND LINE
ALTERS ?  SECOND <TEST >              (CTRL/A) typed - '<' echoed.
      2   SECOND TEST LINE            (CR) typed - '>' echoed.
ALTERS ?
```

Typing a (CTRL/Q) aborts the current line modification.

Typing a (CTRL/D) truncates the line.

Example:

```
        ? MD 2 (CR)                   Modify line 2
      2   SECOND TEST LINE
ALTERS ?  SECOND TEST (CTRL/D)        (CTRL/D)truncates rest of line.
      2   SECOND TEST
```

5.4.9 MS - Modify String

The MS command modifies lines like the MD command. The argument must be a character string enclosed in single quotes ('). If the string argument is not found in the text, EDIT16 merely prompts for the next command.

Example:

```
      ?  MS 'SEC' (CR)        'SEC' string is in line 2; thus, proceed to modify line 2.
   2     SECOND LINE
ALTERS?
```

5.4.10 DL - Delete Line

The DL command deletes a line or a range of lines. After the specified lines are deleted, EDIT16 automatically renumbers the text.

Example:

```
?  DL 1,3/5 (CR)        Delete lines 1, 3, 4, and 5.
```

EDIT16 types 'VOID RANGE' when the specified line is out of range.

5.4.11 CL - Copy Line

The CL command copies existing lines to anywhere in the buffer. Lines are renumbered automatically.

Example:

```
?  CL 1/2 (CR)          Append copies of lines 1 and 2 to the buffer.

?  CL 3/4 TO 6 (CR)     Insert copies of lines 3 and 4 before line 6. Line 6
                        becomes line 8.
```

5.4.12 MV - Move Line

The MV command moves lines to anywhere in the buffer. Note that MV is like the CL command except that the lines being moved are deleted from their original locations. Lines are renumbered automatically.

Example:

```
?  MV 1 (CR)            This command results in line 1 becoming the last
                        line and line 2 then becomes the first line - and on,
                        through the buffer.
```

5.4.13 CB - Clear Buffer

The CB command clears the entire buffer. If an output command is issued and the buffer is clear, EDIT16 types:

NO ACTIVE FILE

5.4.14 FS - Find String

The FS command searches the entire edit buffer for all occurrences of the specified string. Lines containing the string are typed. If no string is found EDIT16 types:

VOID RANGE

It then prompts for the next command. Quote mark (') is treated just like any other character.

Example:

? FS 'AC1' CR	Find string 'AC1' in edit buffer.
VOID RANGE	No string 'AC1' in edit buffer.
? FS 'X'FF' CR	Find string X'FF in edit buffer.
3 .WORD X'FF	FF found in line 3.
?	

5.4.15 ST - Set Tab

The ST command enables fixed horizontal spacing when collecting text in keyboard mode.

Example:

```
            ?  ST  CR
                          1 2 3 4 5 6 7 8 9 % 1 2 3 4....% 1 2 3 4 5
START?                          1               1            1
VERIFY?                         1               1            1
                           Column 1                              Column 65
```

EDIT16 prints the row of numbers, one character for each column. After 65 columns are marked in this manner, EDIT16 prints START?. The user then can mark up to three columns, where a tab is desired, with any printable character (in the above example, the number 1 is used). EDIT16 replies with a verification by typing a corresponding 1 in each position where a tab is specified.

5.5 OPERATING PROCEDURES

5.5.1 Starting

The EDIT16 program may be loaded by the absolute loader or general loader and starts automatically. Once loaded, the following sequence occurs:

NSC EDIT16 REV X
MEMORY:

Type in the memory range in the format 0:xx, where xx lies in the range $4 \leq xx \leq 64$ and must be a multiple of 4. "xx" must not be greater than the actual system memory. To get the default of 4, type CR . EDIT16 repeats the message if input format is erroneous. If accepted, EDIT16 goes into command mode and types the prompt character (?).

5.5.2 Error Corrections

EDIT16 processing may be interrupted by the use of the following:

1. CTRL/Q aborts the current command input if in command mode, or the current line input if in text mode.

2. Pressing any key on the keyboard during an input or output operation aborts processing and EDIT16 goes into command mode.

3. Back arrow ←, typed while in text mode, deletes the previous character input. Successive back arrows delete preceding characters in the line. Back arrows may not be used in command mode or when inserting characters (characters typed after CNTL/A) in the modify line/string commands.

5.6 SAMPLE OF EDIT16 USE

The program, listed in figure 5-1 needs to be corrected; the corrections are shown pencilled-in. Figure 5-2 shows the use of EDIT16 commands to implement changes to the sample program. Figure 5-3 shows the corrected program listing.

```
        .TITLE 'MULTIPLY AND DIVIDE ROUTINES'
;
;

? LS
  PSIGN=2                        ; AC0=POS   JUMP CONDITION
  NRGT0=11                       ; AC0<=0   JUMP CONDITION
  SEL=2   SELFF                  ; SELX FLAG
  BIT0=3                         ; BIT0=1   JUMP CONDITION
  AC0=0                          ; DEFINE ACCUMULATORS
  AC1=1
  AC2=2
  AC3=3
;
;          MAIN CALLING PROGRAM
;
          LD       AC0,EA        ; LOAD MULTIPLIER
          JSR      MULT          ; CALL MULTIPLY ROUTINE
          HALT
          JMP      .-3           ; RERUN
          ST       AC3,SAV3      ; SAVE AC3
          LD       AC3,EA        ; LOAD DIVISOR
          JSR      DIVD          ; CALL DIVIDE ROUTINE
          HALT
          JMP      .-4           ; RERUN
EA:       .WORD    0
;
;          SUBROUTINE MULTIPLY
;
MULT:     ST       AC2,SAVE2     ; SAVE AC2
          ST       AC3,SAVE3     ; SAVE AC3
          LI       AC2,0 2       ; CLEAR AC2
          LI       AC3,~~0~~16   ; BIT COUNT=16
          CAI      AC0,0         ; COMPLIMENT AC0 TO SIMPLIFY
                                 ; BRANCHING ON MULTIPLIER BIT0
          SFLG     SELFF         ; INCLUDE LINK IN SHIFTS
          SHL      AC2,1         ; CLEAR LINK
LOOP:     BOC      BIT0,.+2      ; BRANCH IF AC0 COMPLIMENTED=0
          RADD     AC1,AC~~1~~2  ; AC1+AC2 --> AC2
          ROR      AC2,1         ; ROTATE RESULT OF ADD INTO LINK
          SHR      AC0,1         ; SHIFT LINK INTO AC0
          AISZ     AC3,-1        ; DECR COUNT, SKIP IF ZERO
          ~~JSR~~ JMP  LOOP
          RCPY     AC0,AC1       ; MOVE LO ORDER RESULT TO AC1
          RCPY     AC2,AC0       ; MOVE HI ORDER RESULT TO AC0
          LD       AC3,SAVE3     ; RESTORE AC3
          LD       AC2,SAVE2     ; RESTORE AC2
          PFLG     SELFF         ; CLEAR SELF
          RTS
SAVE2:    ~~.=.+1~~  .WORD 0
SAVE3:    ~~.=.+1~~  .WORD 0
;
```

Figure 5-1. Sample Program Needing Correction (Sheet 1 of 2)

```
;            SUBROUTINE DIVD DIVIDE
;
COUNT:       .WORD      0
DIVD:        ST         AC2,SAV2         ; SAVE AC2
             RCPY       AC0,AC2
             CAI        AC0,1
             RADD       AC3,AC0          ; SUBTRACT HI ORDER FROM DIVISOR
             BOC        NRGT0,OVFLW      ; IS HI ORDER 7= DIVISOR
             LI         AC0,-16          ; NO
             ST         AC0,COUNT        ; SET COUNT = 16
             SFLG       SELFF            ; SET SELX
             LI         AC0,0
             SHL        AC0,1            ; CLEAR LINK
             SHL        AC1,1
POOL:        ROL        AC2,1            ; ROTATE HI ORDER LEFT WITH LINK
             RCPY       AC2,AC0.
             CAI        AC0,1
             RADD       AC3,AC0          ; SUBTRACT HI ORDER FROM DIVISOR
             BOC        NRGT0,GOES       ; IS HI ORDER >= DIVISOR
             LI         AC0,0            ; NO
             SHL        AC0,1            ; CLEAR LINK
             JMP        SHFTLO
GOES:        CAI        AC0,1            ; YES
             RCPY       AC0,AC2          ; HI ORDER = HI ORDER - DIVISOR
             LI         AC0,-1
             SHL        AC0,1            ; SET LINK
SHFTLO:      ROL        AC1,1            ; ROTATE LO ORDER WITH LINK LEFT
             ISZ        COUNT            ; ARE WE DONE
             JMP        POOL             ; NO
             RCPY       AC1,AC0          ; YES
             BOC        PSIGN,*+2        ; IS RESULT NEG
             JMP        OVFLW            ; YES, OVERFLOW
             RCPY       AC2,AC0          ; NO MOVE REMAINDER TO AC0,QUOTE
                                         ; IN AC1
OVFLW:       LD         AC3,H7000
             RADD       AC3,AC3          ; SET OVERFLOW
             JMP        DONE
SAV2:        .WORD      0
SAV3:        .WORD      0
H7000:       .WORD      X'7000
.END

?
```

```
DONE:        PFLG       SELFF            ; CLEAR SELX
             LD         AC2,SAV2         ; RESTORE AC2
             LD         AC3,SAV3         ; RESTORE AC3
             RTS
```

Figure 5-1. Sample Program Needing Correction (Sheet 2 of 2)

```
   ?  KB TO 1
->         .TITLE  'MULTIPLY AND DIVIDE ROUTINES' CR
->   ; CR
->   ; CR
->  CTRL/Q ***

   ?  MD 4/11 CR
   4    PSIGN=2                                  ; AC0=POS   JUMP CONDITION
ALTERS?PSIGN    =          2 CR

   4    PSIGN   =          2                     ; AC0=POS   JUMP CONDITION
ALTERS? CR
   5    NRGT0=11                                 ; AC0<=0   JUMP CONDITION
ALTERS?NRGT0    =          11 CR

   5    NRGT0   =          11                    ; AC0<=0   JUMP CONDITION
ALTERS? CR
   6    SEL=2                                    ; SELX FLAG
ALTERS?SEL      =          2 CR

   6    SEL     =          2                     ; SELX FLAG
ALTERS? CR
   7    BIT0=3                                   ; BIT0=1   JUMP CONDITION
ALTERS?BIT0     =          3 CR

   7    BIT0    =          3                     ; BIT0=1   JUMP CONDITION
ALTERS? CR
   8    AC0=0                                    ; DEFINE ACCUMULATORS
ALTERS?AC0      =          0 CR

   8    AC0     =          0                     ; DEFINE ACCUMULATORS
ALTERS? CR
   9    AC1=1
ALTERS?AC1      =          1 CR

   9    AC1     =          1
ALTERS? CR
  10    AC2=2
ALTERS?AC2      =          2 CR

  10    AC2     =          2
ALTERS? CR
  11    AC3=3
ALTERS?AC3      =          3 CR

  11    AC3     =          3
ALTERS? CR

   ?
```

Figure 5-2. EDIT16 - Implementation of Correction Commands (Sheet 1 of 4)

```
  ?  MS 'MAIN' (CR)
 13    ;              MAIN PROGRAM
ALTERS?;              MAIN <CALLING >

 13    ;              MAIN CALLING PROGRAM
ALTERS? (CR)
 83                   RCPY     AC2,AC0          ; NO MOVE REMAINDER TO AC0,QUOTE
ALTERS?               RCPY     AC2,AC0          ; NO MOVE REMAINDER TO AC0,QUOT↑ (CR)

 83                   RCPY     AC2,AC0          ; NO MOVE REMAINDER TO AC0,QUOT
ALTERS? (CR)
  ?  MD 16,21,26,31,34,37,41 (CR)
 16                   JSR      MULT
ALTERS?               JSR      MULT            ; CALL MULTIPLY ROUTINE (CR)

 16                   JSR      MULT            ; CALL MULTIPLY ROUTINE
ALTERS? (CR)
 21                   JSR      DIVD
ALTERS?               JSR      DIVD            ; CALL DIVIDE ROUTINE (CR)

 21                   JSR      DIVD            ; CALL DIVIDE ROUTINE
ALTERS? (CR)
 26    ;              SUBROUTINE MULT
ALTERS?;              SUBROUTINE MULTIPLY (CR)

 26    ;              SUBROUTINE MULTIPLY
ALTERS? (CR)
 31                   LI       AC3,016          ; BIT COUNT=16
ALTERS?               LI       AC3,↑ (CR)

 31                   LI       AC3,16          ; BIT COUNT=16
ALTERS?               LI       AC3,16< >

 31                   LI       AC3,16           ; BIT COUNT=16
ALTERS? (CR)
 34                   SFLG     SEL              ; INCLUDE LINK IN SHIFTS
ALTERS?               SFLG     SELFF (CR)

 34                   SFLG     SELFF            ; INCLUDE LINK IN SHIFTS
ALTERS? (CR)
 37                   RADD     AC1,AC1          ; AC1+AC2 --> AC2
ALTERS?               RADD     AC1,AC2 (CR)

 37                   RADD     AC1,AC2          ; AC1+AC2 --> AC2
ALTERS? (CR)
 41                   JSR      LOOP
ALTERS?               JMP (CR)

 41                   JMP      LOOP
ALTERS? (CR)

  ?
```

Figure 5-2. EDIT16 - Implementation of Correction Commands (Sheet 2 of 4)

```
  ?  MS 'SEL ' (CR)
  6    SEL       =            2                ; SELX FLAG
ALTERS?SELFF (CR)

  6    SELFF     =            2                ; SELX FLAG
ALTERS? (CR)
  61             SFLG      SEL                 ; SET SELX
ALTERS?          SFLG      SELFF (CR)

  61             SFLG      SELFF               ; SET SELX
ALTERS? (CR)
  ?  MD 48/49,51 (CR)
  48   SAVE2:    .=.+1
ALTERS?SAVE2:    .WORD     0 (CR)

  48   SAVE2:    .WORD     0
ALTERS? (CR)
  49   SAVE3:    .=.+1
ALTERS?SAVE3:    .WORD (CR)

  49   SAVE3:    .WORD
ALTERS?SAVE3:    .WORD     0 (CR)

  49   SAVE3:    .WORD     0
ALTERS? (CR)
  51   ;         SUBROUTINE DIVD
ALTERS?;         SUBROUTINE DIVIDE (CR)

  51   ;         SUBROUTINE DIVIDE
ALTERS? (CR)
  ?  LS 53,94 (CR)
  53   COUNT:    .WORD     0
VOID RANGE

  ?  LS 91 (CR)
  91   .END

  ?  MV 53 TO 91 (CR)

  ?  MS 'GOS:' (CR)
  72   GOS:      CAI       AC0,1               ; YES
ALTERS?GO<E>

  72   GOES:     CAI       AC0,1               ; YES
ALTERS?GOES:↑ (CR)

  72   GOES:     CAI       AC0,1               ; YES
ALTERS? (CR)
  ?  FS 'PSIGN' (CR)
  4    PSIGN     =            2                ; AC0=POS  JUMP CONDITION
  80             BOC       PSIGN,+2            ; IS RESULT NEG

  ?
```

Figure 5-2. EDIT16 - Implementation of Correction Commands (Sheet 3 of 4)

```
  ?  MD 8Ø (CR)
 8Ø                BOC        PSIGN,+2               ; IS RESULT NEG
ALTERS?            BOC        PSIGN,.+2 (CR)

 8Ø                BOC        PSIGN,.+2              ; IS RESULT NEG
ALTERS? (CR)

  ?  FS 'OVFLW' (CR)
 57                BOC        NRGTØ,OVFLW            ; IS HI ORDER 7= DIVISOR
 81                JMP        OVFLW                  ; YES, OVERFLOW
 84    OVFLW:      LD         AC3,H7ØØØ

  ?  KB TO 84 (CR)
->  DONE:     PFLG       SELFF               ; CLEAR SELX (CR)
->            RA (CTRL/Q) ***
->            LD         AC2,SAV2            ; RESTORE AC2 (CR)
->            LD         AC3,SAV3            ; RESTORE AC3 (CR)
->            RTS                                          (CR)
-> (CTRL/Q) ***

  ?  LL (CR)
 95    .END

  ?  MD 95 (CR)
 95    .END
ALTERS?<          >

 95                .END
ALTERS? (CR)

  ?
```

Figure 5-2. EDIT16 - Implementation of Correction Commands (Sheet 4 of 4)

```
?   LS
                .TITLE  'MULTIPLY AND DIVIDE ROUTINES'
    ;
    ;
    PSIGN       =       2               ; ACØ=POS  JUMP CONDITION
    NRGTØ       =       11              ; ACØ<=Ø  JUMP CONDITION
    SELFF       =       2               ; SELX FLAG
    BITØ        =       3               ; BITØ=1  JUMP CONDITION
    ACØ         =       Ø               ; DEFINE ACCUMULATORS
    AC1         =       1
    AC2         =       2
    AC3         =       3
    ;
    ;           MAIN CALLING PROGRAM
    ;
                LD      ACØ,EA          ; LOAD MULTIPLIER
                JSR     MULT            ; CALL MULTIPLY ROUTINE
                HALT
                JMP     .-3             ; RERUN
                ST      AC3,SAV3        ; SAVE AC3
                LD      AC3,EA          ; LOAD DIVISOR
                JSR     DIVD            ; CALL DIVIDE ROUTINE
                HALT
                JMP     .-4             ; RERUN
    EA:         .WORD   Ø
    ;
    ;           SUBROUTINE MULTIPLY
    ;
    MULT:       ST      AC2,SAVE2       ; SAVE AC2
                ST      AC3,SAVE3       ; SAVE AC3
                LI      AC2,Ø           ; CLEAR AC2
                LI      AC3,16          ; BIT COUNT=16
                CAI     ACØ,Ø           ; COMPLIMENT ACØ TO SIMPLIFY
                                        ; BRANCHING ON MULTIPLIER BITØ
                SFLG    SELFF           ; INCLUDE LINK IN SHIFTS
                SHL     AC2,1           ; CLEAR LINK
    LOOP:       BOC     BITØ,.+2        ; BRANCH IF ACØ COMPLIMENTED=Ø
                RADD    AC1,AC2         ; AC1+AC2 --> AC2
                ROR     AC2,1           ; ROTATE RESULT OF ADD INTO LINK
                SHR     ACØ,1           ; SHIFT LINK INTO ACØ
                AISZ    AC3,-1          ; DECR COUNT, SKIP IF ZERO
                JMP     LOOP
                RCPY    ACØ,AC1         ; MOVE LO ORDER RESULT TO AC1
                RCPY    AC2,ACØ         ; MOVE HI ORDER RESULT TO ACØ
                LD      AC3,SAVE3       ; RESTORE AC3
                LD      AC2,SAVE2       ; RESTORE AC2
                PFLG    SELFF           ; CLEAR SELF
                RTS
    SAVE2:      .WORD   Ø
    SAVE3:      .WORD   Ø
    ;
```

Figure 5-3. Corrected Program Listing (Sheet 1 of 2)

```
;          SUBROUTINE DIVIDE
;
DIVD:      ST        AC2,SAV2           ; SAVE AC2
           RCPY      ACØ,AC2
           CAI       ACØ,1
           RADD      AC3,ACØ            ; SUBTRACT HI ORDER FROM DIVISOR
           BOC       NRGTØ,OVFLW        ; IS HI ORDER 7= DIVISOR
           LI        ACØ,-16            ; NO
           ST        ACØ,COUNT          ; SET COUNT = 16
           SFLG      SELFF              ; SET SELX
           LI        ACØ,Ø
           SHL       ACØ,1              ; CLEAR LINK
           SHL       AC1,1
POOL:      ROL       AC2,1              ; ROTATE HI ORDER LEFT WITH LINK
           RCPY      AC2,ACØ
           CAI       ACØ,1
           RADD      AC3,ACØ            ; SUBTRACT HI ORDER FROM DIVISOR
           BOC       NRGTØ,GOES         ; IS HI ORDER >= DIVISOR
           LI        ACØ,Ø              ; NO
           SHL       ACØ,1              ; CLEAR LINK
           JMP       SHFTLO
GOES:      CAI       ACØ,1              ; YES
           RCPY      ACØ,AC2            ; HI ORDER = HI ORDER - DIVISOR
           LI        ACØ,-1
           SHL       ACØ,1              ; SET LINK
SHFTLO:    ROL       AC1,1              ; ROTATE LO ORDER WITH LINK LEFT
           ISZ       COUNT              ; ARE WE DONE
           JMP       POOL               ; NO
           RCPY      AC1,ACØ            ; YES
           BOC       PSIGN,.+2          ; IS RESULT NEG
           JMP       OVFLW              ; YES, OVERFLOW
           RCPY      AC2,ACØ            ; NO MOVE REMAINDER TO ACØ,QUOT
                                        ; IN AC1
DONE:      PFLG      SELFF              ; CLEAR SELX
           LD        AC2,SAV2           ; RESTORE AC2
           LD        AC3,SAV3           ; RESTORE AC3
           RTS
OVFLW:     LD        AC3,H7ØØØ
           RADD      AC3,AC3            ; SET OVERFLOW
           JMP       DONE
SAV2:      .WORD     Ø
SAV3:      .WORD     Ø
H7ØØØ:     .WORD     X'7ØØØ
COUNT:     .WORD     Ø
           .END

?
```

Figure 5-3. Corrected Program Listing (Sheet 2 of 2)

Punch the edited text on paper tape. Observe that punched text is echoed on the printer.

```
   ?  PT
TURN PUNCH ON
        .TITLE   'MULTIPLY AND DIVIDE ROUTINES'
;
;
PSIGN    =        2              ; AC0=POS   JUMP CONDITION
NRGT0    =        11             ; AC0<=0   JUMP CONDITION
SELFF    =        2              ; SELX FLAG
BIT0     =        3              ; BIT0=1   JUMP CONDITION
AC0      =        0              ; DEFINE ACCUMULATORS
AC1      =        1
AC2      =        2
AC3      =        3
;
;       MAIN CALLING PROGRAM
;
        LD       AC0,EA          ; LOAD MULTIPLIER
```

Figure 5-4. Program Listing on PT Command

Appendix A

IMP-16 CHARACTER SET

Table A-1. IMP-16 Character Set

Character		7-Bit Hexadecimal Number	Punched Card Code	Character		7-Bit Hexadecimal Number	Punched Card Code
ASCII	029			ASCII	029		
NUL		00	12-0-1-8-9	!		21	11-2-8
SOH		01	12-1-9	"		22	7-8
STX		02	12-2-9	#		23	3-8
ETX		03	12-3-9	$		24	11-3-8
EOT		04	7-9	%		25	0-4-8
ENQ		05	0-5-8-9	&		26	12
ACK		06	0-6-8-9	'		27	5-8
BEL		07	0-7-8-9	(		28	12-5-8
BS		08	11-6-9	)		29	11-5-8
HT		09	12-5-9	*		2A	11-4-8
LF		0A	0-5-9	+		2B	12-6-8
VT		0B	12-3-8-9	,		2C	0-3-8
FF		0C	12-4-8-9	-		2D	11
CR		0D	12-5-8-9	.		2E	12-3-8
SO		0E	12-6-8-9	/		2F	0-1
SI		0F	12-7-8-9	0		30	0
DLE		10	12-11-1-8-9	1		31	1
DC1		11	11-1-9	2		32	2
DC2		12	11-2-9	3		33	3
DC3		13	11-3-9	4		34	4
DC4		14	4-8-9	5		35	5
NAK		15	5-8-9	6		36	6
SYN		16	2-9	7		37	7
ETB		17	0-6-9	8		38	8
CAN		18	11-8-9	9		39	9
EM		19	11-1-8-9	:		3A	2-8
SUB		1A	7-8-9	;		3B	11-6-8
ESC		1B	0-7-9	<		3C	12-4-8
FS		1C	11-4-8-9	=		3D	6-8
GS		1D	11-5-8-9	>		3E	0-6-8
RS		1E	11-6-8-9	?		3F	0-7-8
US		1F	11-7-8-9	@		40	4-8
SP		20	No Punches				
A		41	12-1	a		61	12-0-1
B		42	12-2	b		62	12-0-2
C		43	12-3	c		63	12-0-3
D		44	12-4	d		64	12-0-4
E		45	12-5	e		65	12-0-5
F		46	12-6	f		66	12-0-6
G		47	12-7	g		67	12-0-7
H		48	12-8	h		68	12-0-8
I		49	12-9	i		69	12-0-9
J		4A	11-1	j		6A	12-11-1
K		4B	11-2	k		6B	12-11-2

Table A-1. IMP-16 Character Set (Cont)

Character		7-Bit Hexadecimal Number	Punched Card Code	Character		7-Bit Hexadecimal Number	Punched Card Code
ASCII	029			ASCII	029		
L		4C	11-3	l		6C	12-11-3
M		4D	11-4	m		6D	12-11-4
N		4E	11-5	n		6E	12-11-5
O		4F	11-6	o		6F	12-11-6
P		50	11-7	p		70	12-11-7
Q		51	11-8	q		71	12-11-8
R		52	11-9	r		72	12-11-9
S		53	0-2	s		73	11-0-2
T		54	0-3	t		74	11-0-3
U		55	0-4	u		75	11-0-4
V		56	0-5	v		76	11-0-5
W		57	0-6	w		77	11-0-6
X		58	0-7	x		78	11-0-7
Y		59	0-8	y		79	11-0-8
Z		5A	0-9	z		7A	11-0-9
[	¢	5B	12-2-8			7B	12-0
\	0-8-2	5C	0-8-2			7C	12-11
]	|	5D	12-7-8	ALT		7D	11-0
↑	¬	5E	11-7-8	ESC		7E	11-0-1
←	_	5F	0-5-8	DEL, RUB		7F	12-7-9
`		60	8-1				

Table A-2. Legend for Nonprintable Characters

Character	Definition	Character	Definition
NUL	Null	SO	Shift out
SOH	Start of heading (also start of message)	SI	Shift in
STX	Start of text (also EOA, end of address)	DLE	Data link escape
		DC1	Device control 1
ETX	End of text (also EOM, end of message)	DC2	Device control 2
		DC3	Device control 3
EOT	End of transmission (also END)	DC4	Device control 4
		NAK	Negative acknowledge
ENQ	Enquiry (also ENQRY, WRU)	SYN	Synchronous idle (SYNC)
ACK	Acknowledge (also RU)	ETB	End of transmission block
BEL	Rings the bell	CAN	Cancel (CANCL)
BS	Backspace	EM	End of medium
HT	Horizontal tab	SUB	Substitute
LF	Line feed or line space (also new line, advances paper to next line, beginning of line)	ESC	Escape. Prefix
		FS	File Separator
		GS	Group separator
VT	Vertical tab (VTAB)	RS	Record separator
FF	Form feed to top of next page (PAGE)	US	Unit separator
CR	Carriage return	SP	Space

Appendix B

INSERTION OF RLM CORRECTIONS

Corrections may be inserted in RLMs (decks or tapes) by correcting DATA records or adding DATA records just before the RLM END record. These records should agree in format with a standard DATA record (see IMP-16 Programming and Assembler Manual, appendix A). For simplicity, this section explains how a single record may be used to load a single memory location.

The first word of the corrector record should contain X'8005. The second word of the corrector record should contain a checksum word X'0000 so GENLDR does not attempt to checksum the record. The third word of the record should contain one of the following values indicating the relocation, if any is to be performed:

X'0000	Absolute record (no relocation)
X'0001	Base sector relocatable record
X'0002	Top sector relocatable record

Word 4 of the record should contain either the absolute initial load address of the record or the proper displacement relative to the base-sector or the top-sector origin of the RLM (as indicated by word 3).

Word 5 of the record should contain one of the following values to indicate the relocation, if any, to be performed upon the contents of the data word:

X'0000	The data is absolute.
X'4000	The data references a base-sector relocatable address.
X'8000	The data references a top-sector relocatable address.

Word 6 should contain X'0000.

The data word to be inserted should appear in word 7.

Example:

In the top sector of an RLM, the following correction is to be inserted:

Location	Value	Relocation
DAA	8109	B
DAB	5802	A
DAC	A107	T
DAD	6107	B
DAE	A200	A
DAF	4A01	A
DB0	2435	B

The origin of the current top sector is D00. The correctors should contain:

Columns

```
1       5       9       13      17      21      25

8 0 0 5 0 0 0 0 0 0 0 2 0 0 A A 4 0 0 0 0 0 0 0 8 1 0 9
8 0 0 5 0 0 0 0 0 0 0 2 0 0 A B 0 0 0 0 0 0 0 0 5 8 0 2
8 0 0 5 0 0 0 0 0 0 0 2 0 0 A C 8 0 0 0 0 0 0 0 A 1 0 7
8 0 0 5 0 0 0 0 0 0 0 2 0 0 A D 4 0 0 0 0 0 0 0 6 1 0 7
8 0 0 5 0 0 0 0 0 0 0 2 0 0 A E 0 0 0 0 0 0 0 0 A 2 0 0
8 0 0 5 0 0 0 0 0 0 0 2 0 0 A F 0 0 0 0 0 0 0 0 4 A 0 1
8 0 0 5 0 0 0 0 0 0 0 2 0 0 B 0 4 0 0 0 0 0 0 0 2 4 3 5
```

Appendix C

FORMAT OF INSTRUCTIONS

A summary of the instruction types and their assembler language formats is given below for reference. A more-detailed breakdown of the instruction codes is shown in the next table; it is suitable for hand-coding small programs.

Instruction Type	MACHINE FORMAT	Assembler Language Format		Remarks
Register to Register	[op][sr][dr][op][not used][op]	Op	sr, dr	
Register to Memory	[op][r][disp]	Op	r, disp	
Memory Reference (Class 1)	[op][r][xr][disp]	Op	r, disp(xr)	Direct
		Op	r, @disp(xr)	Indirect
Memory Reference (Class 2)	[op][xr][disp]	Op	disp(xr)	Direct
		Op	@disp(xr)	Indirect
I/O and Miscellaneous	[op][ctl]	Op	ctl	
Branch	[op][cc][disp]	Op	cc, disp	

Explanation of Symbols

OP	Instruction Mnemonic	disp	Displacement Value
Op	Operation Code	cc	Condition Code Value
sr	Source Register Vaoue	r	Register Value
dr	Destination Register Value	ctl	Control Bits Value
xr	Index Register Value		

Table C-1. Instruction Set with Bit Patterns

Mnemonic	Base
LD	8000
LD Indirect	9000
ST	A000
ST Indirect	B000
ADD	C000
SUB	D000
SKG	E000
SKNE	F000

Word Format
1 = BASE ∨ r ∨ xr ∨ disp

r	REGISTER
0000	0
0400	1
0800	2
0C00	3

xr	ADDRESSING TECHNIQUE
0000	BASE SECTOR
0411	PC RELATIVE
0200	INDEXED - AC2
0300	INDEXED - AC3

AND	6000
OR	6800
SKAZ	7000

r	REGISTER
0000	0
0400	1

ISZ	7800	JMP Indirect	2400
DSZ	7C00	JSR	2800
JMP	2000	JSR Indirect	2C00

Word Format
1 = BASE ∨ cc ∨ disp

BOC 1000

Branch on	INT	AC0=0	AC0 ≥ 0	AC0 ODD	AC0 Bit 1=1	AC≠0	CPINT	START
CC	0000	0100	0200	0300	0400	0500	0600	0700
Branch on	**STFL**	**INEN**	**CYOV**	**AC0 ≤ 0**	**USER / POA**	**USER / SEL**	**USER**	**USER**
CC	0800	0900	0A00	0B00	0C00	0D00	0E00	0F00

16C / 16L

Word Format
1 = BASE ∨ r ∨ disp

AISZ	4800	
LI	4C00	
CAI	5000	
PUSH	4000	
PULL	4400	
XCHRS	5400	
ROR/ROL	5800	LEFT DISP POSITIVE
SHR/SHL	5C00	RIGHT DISP NEGATIVE

r	REGISTER
0000	0
0100	1
0200	2
0300	3

Word Format
1 = BASE ∨ sr ∨ dr

RADD	3000
RXCH	3080
RCPY	3081
RXOR	3082
RAND	3083

sr	dr	REGISTER
0000	0000	0
0400	0100	1
0800	0200	2
0C00	0300	3

Table C-1. Instruction Set with Bit Patterns (Continued)

Mnemonic	Base	fc	FLAG			Word Format
						1 = BASE ∨ fc ∨ ctl
SFLG	0800	0000	8			
PFLG	0880	0100	9			
		0200	10			
		0300	11			
		0400	12			
		0500	13			
		0600	14			
		0700	15			
						Word Format
						1 = BASE ∨ ctl
HALT	0000	RTI	0100	RIN	0400	
PUSHF	0080	RTS	0200	ROUT	0600	
PULLF	0280	JSRI	0380			

The instruction is formed by the inclusive Or of each field. For example, the instruction RADD 2, 3 is coded as X'3C00.

For instructions that use the CTL field, only the first 7 bits (bits 0 through 6) are considered.

Examples of coding follow:

Example 1	Comments
RADD 2, 3	Add AC2 to AC3.

BASE = 3000
sr = 0800
dr = 0300
INSTRUCTION = 3C00

Example 2	Comments
JMP-1 (3)	Jump to the location specified by the index register AC3 modified by the displacement-1.

BASE = 2000
xr = 0300
disp = 00FF
INSTRUCTION = 23FF

Example 3	Comments
SHR 0, 1	Shift the contents of AC0 one place to the right.

BASE = 5C00
r = 0000
disp = 00FF
INSTRUCTION = 5CFF

Appendix D

CONVERSION TABLES

Table D-1. Positive Powers of Two

n	2^n	n	2^n
1	2	51	22517 99813 68524 8
2	4	52	45035 99627 37049 6
3	8	53	90071 99254 74099 2
4	16	54	18014 39850 94819 84
5	32	55	36028 79701 89639 68
6	64	56	72057 59403 79279 36
7	128	57	14411 51880 75855 872
8	256	58	28823 03761 51711 744
9	512	59	57646 07523 03423 488
10	1024	60	11529 21504 60684 6976
11	2048	61	23058 43009 21369 3952
12	4096	62	46116 86018 42738 7904
13	8192	63	92233 72036 85477 5808
14	16384	64	18446 74407 37095 51616
15	32768	65	36893 48814 74191 03232
16	65536	66	73786 97629 48382 06464
17	13107 2	67	14757 39525 89676 41292 8
18	26214 4	68	29514 79051 79352 82585 6
19	52428 8	69	59029 58103 58705 65171 2
20	10485 76	70	11805 91620 71741 13034 24
21	20971 52	71	23611 83241 43482 26068 48
22	41943 04	72	47223 66482 86964 52136 96
23	83886 08	73	94447 32965 73929 04273 92
24	16777 216	74	18889 46593 14785 80854 784
25	33554 432	75	37778 93186 29571 61709 568
26	67108 864	76	75557 86372 59143 23419 136
27	13421 7728	77	15111 57274 51828 64683 8272
28	26843 5456	78	30223 14549 03657 29367 6544
29	53687 0912	79	60446 29098 07314 58735 3088
30	10737 41824	80	12089 25819 61462 91747 06176
31	21474 83648	81	24178 51639 22925 83494 12352
32	42949 67296	82	48357 03278 45851 66988 24704
33	85899 34592	83	96714 06556 91703 33976 49408
34	17179 86918 4	84	19342 81311 38340 66795 29881 6
35	34359 73836 8	85	38685 62622 76681 33590 59763 2
36	68719 47673 6	86	77371 25245 53362 67181 19526 4
37	13743 89534 72	87	15474 25049 10672 53436 23905 28
38	27487 79069 44	88	30948 50098 21345 06872 47810 56
39	54975 58138 88	89	61897 00196 42690 13744 95621 12
40	10995 11627 776	90	12379 40039 28538 02748 99124 224
41	21990 23255 552	91	24758 80078 57076 05497 98248 448
42	43980 46511 104	92	49517 60157 14152 10995 96496 896
43	87960 93022 208	93	99035 20314 28304 21991 92993 792
44	17592 18604 4416	94	19807 04062 85660 84398 38598 7584
45	35184 37208 8832	95	39614 08125 71321 68796 77197 5168
46	70368 74417 7664	96	79228 16251 42643 37593 54395 0336
47	14073 74883 55328	97	15845 63250 28528 67518 70879 00672
48	28147 49767 10656	98	31691 26500 57057 35037 41758 01344
49	56294 99534 21312	99	63382 53001 14114 70074 83516 02688
50	11258 99906 84262 4	100	12676 50600 22822 94014 96703 20537 6
		101	25353 01200 45645 88029 93406 41075 2

Table D-2. Negative Powers of Two

n	2^{-n}
0	1.0
1	0.5
2	0.25
3	0.125
4	0.0625
5	0.03125
6	0.01562 5
7	0.00781 25
8	0.00390 625
9	0.00195 3125
10	0.00097 65625
11	0.00048 82812 5
12	0.00024 41406 25
13	0.00012 20703 125
14	0.00006 10351 5625
15	0.00003 05175 78125
16	0.00001 52587 89062 5
17	0.00000 76293 94531 25
18	0.00000 38146 97265 625
19	0.00000 19073 48632 8125
20	0.00000 09536 74316 40625
21	0.00000 04768 37158 20312 5
22	0.00000 02384 18579 10156 25
23	0.00000 01192 09289 55078 125
24	0.00000 00596 04644 77539 0625
25	0.00000 00298 02322 38769 53125
26	0.00000 00149 01161 19384 76562 5
27	0.00000 00074 50580 59692 38281 25
28	0.00000 00037 25290 29846 19140 625
29	0.00000 00018 62645 14923 09570 3125
30	0.00000 00009 31322 57461 54785 15625
31	0.00000 00004 65661 28730 77392 57812 5
32	0.00000 00002 32830 64365 38696 28906 25
33	0.00000 00001 16415 32182 69348 14453 125
34	0.00000 00000 58207 66091 34674 07226 5625
35	0.00000 00000 29103 83045 67337 03613 28125
36	0.00000 00000 14551 91522 83668 51806 64062 5
37	0.00000 00000 07275 95761 41834 25903 32031 25
38	0.00000 00000 03637 97880 70917 12951 66015 625
39	0.00000 00000 01818 98940 35458 56475 83007 8125
40	0.00000 00000 00909 49470 17729 28237 91503 90625
41	0.00000 00000 00454 74735 08864 64118 95751 95312 5
42	0.00000 00000 00227 37367 54432 32059 47875 97656 25
43	0.00000 00000 00113 68683 77216 16029 73937 98828 125
44	0.00000 00000 00056 84341 88608 08014 86968 99414 0625
45	0.00000 00000 00028 42170 94304 04007 43484 49707 03125
46	0.00000 00000 00014 21085 47152 02003 71742 24853 51562 5
47	0.00000 00000 00007 10542 73576 01001 85871 12426 75781 25
48	0.00000 00000 00003 55271 36788 00500 92935 56213 37890 625
49	0.00000 00000 00001 77635 68394 00250 46467 78106 68945 3125
50	0.00000 00000 00000 88817 84197 00125 23233 89053 34472 65625

Table D-3. Hexadecimal and Decimal Integer Conversion Table

8		7		6		5		4		3		2		1	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
8		7		6		5		4		3		2		1	

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the numbers selected from the table to form the decimal number.

To convert integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

$$
\begin{array}{rrr}
D & = & 13 \\
 & & \times 16 \\ \hline
 & & 208 \\
3 & = & +3 \\ \hline
 & & 211 \\
 & & \times 16 \\ \hline
 & & 3376 \\
4 & = & +4 \\ \hline
 & & 3380
\end{array}
$$

EXAMPLE	
Conversion of Hexadecimal Value	D34
1. D	3328
2. 3	48
3. 4	4
4. Decimal	3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.
   (b) Record the hexadecimal of the column containing the selected number.
   (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).
3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine terms to form the hexadecimal number.

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

		remainder
16	3380	
16	211	4
16	13	3
		D

$3380_{10} = D34_{16}$

EXAMPLE	
Conversion of Decimal Value	3380
1. D	−3328
	52
2. 3	−48
	4
3. 4	−4
4. Hexadecimal	D34

Table D-4. Hexadecimal and Decimal Fraction Conversion Table

1		2			3				4				
Hex	Decimal	Hex	Decimal		Hex	Decimal			Hex	Decimal Equivalent			
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125
.6	.3750	.06	.0234	3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375
1		2			3				4				

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A	in position 1	.6250
Find .0B	in position 2	.0429 6875
Find .00C	in position 3	.0029 2968 7500
.ABC Hex is equal to		.6708 9843 7500

Table D-5. Integer Conversion Table

POWERS OF 16 TABLE

Example: $268,435,456_{10} = (2.68435456 \times 10^8)_{10} = 1000\ 0000_{16} = (10^7)_{16}$

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10 = A
17 592 186 044 416	11 = B
281 474 976 710 656	12 = C
4 503 599 627 370 496	13 = D
72 057 594 037 927 936	14 = E
1 152 921 504 606 846 976	15 = F

$\underbrace{\hphantom{1\ 152\ 921\ 504\ 606\ 846\ 976}}_{\text{Decimal Values}}$

NEGATIVE HEXADECIMAL NUMBERS

The IMP-16 maintains negative numbers in twos-complement form. To convert a number in hexadecimal notation to its twos-complement equivalent, subtract the number from 2^n expressed in hexadecimal form. The number "n" is the number of binary bits in the computer word. For example, if the computer uses a 16-bit word, the number "n" is equal to 16. Thus, the negative of 1245_{16} is derived as follows:

10000	1 0000 0000 0000 0000
-1245	- 0001 0010 0100 0101
EDBB	1110 1101 1011 1011

Note that a hexadecimal number will be negative in the IMP-16 computer if the left most digit is 8, 9, A, B, C, D, E, or F. Thus, FACE is equal to 1111 1010 1100 1110; the twos complement is:

1 0000	1 0000 0000 0000 0000
FACE	1111 1010 1100 1110
532	0000 0101 0011 0010

Appendix E

EDIT16 SYMBOL MEANINGS AND USAGE

The following symbols and their meanings are used in the examples associated with chapter 5, EDIT16.

- All underlined, upper-case characters represent Teletype keyboard, user entries.
- Encircled characters or combinations of characters represent nonprinting, or control characters. If underlined, they represent user initiated input; if not underlined, they represent computer generated characters.
- CTRL/X represents a user input where the Teletype CONTROL key is pressed and held, while the X key is pressed. Similarly, any key (for a printing character) may be pressed in combination with the CONTROL key. The symbol then is CTRL/(key).

Table E-1 lists the symbols used in this text for the EDIT16 program.

Table E-1. EDIT16 Symbols

Symbol	Meaning
←	Back arrow. Indicates Teletype keyboard error correction of previous character, or multiples of characters (depending on number of arrows used.
↑	Echo for CNTL/X input, indicates position of character to be deleted.
→	Prompt, from EDIT16, shows readiness to accept a new line.
?	Prompt, from EDIT16, shows readiness to accept command.
CR	Carriage Return
LF	Line Feed
CTRL/A	Start of character insert operation.
CTRL/D	Truncates the current line.
CTRL/Q	Aborts the current line modification operation.
CTRL/X	Deletes the character in the corresponding position in the previous line.
CTRL/Z	Carriage Tab feature

CHANGE NOTICE NUMBER 1

Publication No. 4200025B
Order No. IMP-16S/025YB

IMP-16 Utilities
Reference Manual

This change is effective immediately for the following programs:

PROMP	4300308B
DEBUG	4300112C
EDIT16	4300332B
STTYIO	4300158C

Page ii. PREFACE. Add to paragraph 1:

To facilitate use of these programs and to minimize loading time, DEBUG, PROMP, and EDIT16 are assembled in absolute format so that they may be loaded into the user's environment using either GENLDR or one of the available absolute loaders.

Page 1-1. Section 1.2. Insert before paragraph 1:

DEBUG is assembled relative to location X'10 in Base Page and location X'210 in Top Sector. This means that DEBUG may be loaded into the user's environment either by an absolute loader (see sections 2.2 and 2.3 of this manual) or by GENLDR (see section 2.5 of this manual). If loaded by an absolute loader, DEBUG will occupy Base Page locations X'10 through X'16, and Top Sector locations X'210 through X'654.

If DEBUG is loaded by GENLDR, DEBUG will appear to occupy Base Page locations 0 through X'F and Top Sector locations 0 through X'20F as well as its normal memory areas. If the GENLDR commands

!OBS 0
!OTS 0

are executed prior to loading DEBUG, the program will occupy the same memory locations as if it were loaded absolutely. If it is desired to use the X'10 locations preceding the DEBUG Base Page or the X'210 locations preceding the DEBUG Top Sector, appropriate !OTS and !OBS commands should be executed. As an example, DEBUG Top Sector might be located at X'1000 by using a

!OTS DF0 (X'1000 - X'210)

command. In essence, the specified Top Sector origin must precede the desired Top Sector origin by X'210 locations.

Page 1-1. Section 1.2. Existing paragraph 1, delete the first sentence, reading:

DEBUG is a relocatable . . . other relocatable program.

Page 1-1. Section 1.2.1, change to read:

The following memory is needed to execute DEBUG:

Top Sector:	X'445 or 1093_{10} words.
Base Page:	7 words.

Page 3-2. Section 3.2.3, sentence 1. Change "STTYI0 (zero)" to read "STTYIO (letter)".

Page 3-2. Section 3.3. Change to read:

STTYIO is assembled as a relocatable load module and must be loaded by GENLDR. (See section 2.5 of this manual.)

Page 4-1. Section 4.1, replace last sentence by:

PROMP output tapes may be used to program either MM5203 2K PROMs in a 256-by-8 structure or MM5204 4K PROMs in a 512-by-8 structure. If MM5203 PROMs are used, two PROMs are required to make up a 256-word by 16-bit memory page.

Page 4-1. Section 4.1, add paragraph 2:

If the BC tape option is selected, PROMP will calculate a 16-bit checksum of the tape and punch it at the end of the tape as four hexadecimal characters in ANSI code.

Page 4-1. Section 4.2.1. Change to read:

PROMP is assembled relative to location X'250 in Top Sector. This means that PROMP may be loaded into the user's environment either by an absolute loader (see sections 2.2 and 2.3 of this manual) or by GENLDR (see section 2.5 of this manual). If loaded by an absolute loader, PROMP will occupy memory locations X'250 through X'9CC.

If PROMP is loaded by GENLDR, it will appear to occupy Top Sector locations 0 through X'24F as well as its normal memory area. If the GENLDR command

    !OTS 0

is executed prior to loading PROMP, the program will occupy the same memory locations as if it were loaded absolutely. If the user desires to use the X'250 locations preceding the PROMP Top Sector, an appropriate !OTS command should be executed. For example, PROMP might be loaded at X'1000 by using a

    !OTS DB0 (X'1000 - X'250)

command. In essence, the specified Top Sector origin must precede the desired Top Sector origin by X'250 locations.

Page 4-1. Section 4.2.2. Change to read:

PROMP requires 780_{16} words of top sector memory.

Page 4-1 through 4-3, section 4.2.3, replace with the following:

4.2.3 Program Messages

To guide the user in the use of PROMP, query messages are typed. There are four general messages corresponding to the four possible program states. Initially, PROMP types:

```
NSC IMP-16 FIRMWARE PAPER TAPE GENERATOR
OUTPUT TYPE:
```

The user must then respond with one of the following codes:

PN - For PN format PROM tape.
BC - For binary (complemented) format PROM tape.
OM - For card RLM to tape RLM conversion.

All responses must be terminated by a carriage return. For the OM option, PROMP types the following message and goes into an output state:

MAKE CARD READER READY
TURN PUNCH ON
HIT ANY KEY TO START

At this point, RLM records are converted from card to paper tape. After processing the END record, PROMP goes into a wait state. The user may now turn off the Teletype punch. Typing any key returns the PROMP program to the initial state.

In either PN or BC options, PROMP goes into an input state. The following message is typed:

INPUT DEVICE:

The user may respond with one of the following codes:

CR - Card Reader
PT - Teletype
ME - Memory

ME Option. If this device is selected, PROMP then types:

SPECIFY MEMORY --

The user must then type the message range, where the user program is located. A range is designated by the start address and the last address, delimited by a colon (:); the range must be in blocks of 256_{10} or 512_{10} words.

Example:

SPECIFY MEMORY -- FF00:FFFF

CR Option. PROMP types the following message:

MAKE CARD READER READY
TO LOAD LM

The card reader must be turned on at this point. The LM will be immediately read into memory.

PT Option. If this device is selected, PROMP types the message:

MAKE TAPE READER READY
TO LOAD LM

The LM will be loaded immediately into memory. After the loading process, when the END record is recognized, PROMP types:

TURN READER OFF

The user must turn off the reader and press any key to initiate the output process.

Mode and Byte Requests. PROMP may be used for punching programming tapes for either the MM5203 2K PROM or the MM5204 4K PROM. After loading the LM, PROMP will query the user as to which PROM is being used.

SET MODE:
TAPE FOR MM5203 2K PROM - TYPE 2
TAPE FOR MM5204 4K PROM - TYPE 4
TYPE:

The MM5203 PROM is structured in a 256-word by 8-bit format, while the MM5204 PROM is structured in a 512-word by 8-bit format. PROMP allows the user to program PROMs for either 256- or 512-word memory regions using the MM5203, and for a 512-word region using the MM5204. In order to do this, the user must respond to the query

BYTE:

with one or more of the options, L, R, LL, LR, HL, and HR according to the following tables:

MM5203 2K PROM

Address	Range	BITS	
		15-8	7-0
N	N+255	LL	LR
N+256	N+511	HL	HR

MM5204 4K PROM

Address	Range	BITS	
		15-8	7-0
N	N+511	L	R

If more than one option is desired, each entry may be separated either by commas or by spaces. For a particular PROM, if all options are desired, the user may respond with a carriage return. After the option is typed, PROMP comes back with the message:

TURN PUNCH ON
HIT ANY KEY TO START

After the output process, PROMP goes into its last state, the wait state. This wait state is provided so that the user can turn off the punch before any message is typed by PROMP. To return to the initial state, the user must press any key on the keyboard.

Typing CTRL/D when PROMP is waiting for a response transfers the user to the initial state. Leaders/trailers are punched automatically. If PN is chosen as an output type code, pressing any key during the output process causes an interrupt of the current output option and initiates processing of the next output option. Under certain conditions, the key must be struck several times in order to cause an interrupt.

Page 4-4.   Section 4.4, following table 4-1, add:

Figure 4-1 illustrates the PROMP operational sequence in flowchart form.

Add figure 4-1 (which is on next page, 4-4A).

Page 5-7.   Section 5.5.1. Insert before paragraph 1:

EDIT16 is assembled relative to location 0 in Base Page and location X'120 in Top Sector. This means that EDIT16 may be loaded either by an absolute loader (see sections 2.2 and 2.3 of this manual) or by GENLDR (see section 2.5 of this manual). If loaded by an absolute loader, EDIT16 will occupy the Base Page locations 0 through X'78, and Top Sector locations X'120 through X'7FB.

If EDIT16 is loaded by GENLDR, it will appear to occupy Top Sector locations 0 through X'11F as well as its normal memory areas. If the GENLDR commands,

```
!OBS 0
!OTS 0
```

are executed prior to loading EDIT16, the program will occupy the same memory locations as if it were loaded absolutely. If it is desired to use the X'120 locations preceding the EDIT16 Top Sector, the appropriate !OTS command should be executed. As an example, EDIT16 Top Sector might be located at X'1000 by using a

```
!OTS EE0 (X'1000 - X'120)
```

command. In essence, the specified top sector origin must precede the desired top sector origin by X'120 locations.

Page 5-7.   Section 5.5.1.   Existing paragraph 1: delete the first sentence, reading:

"The EDIT16 program . . . starts automatically".

Change the second sentence to read:

Once loaded, the following messages are typed:

```
NSC EDIT 16 REV X
MEMORY:
```

Effective immediately, all references to "Relocatable Load Module" (RLM) shall be changed to "Load Module" (LM). Load Modules may be either relocatable or absolute, depending on context or usage.

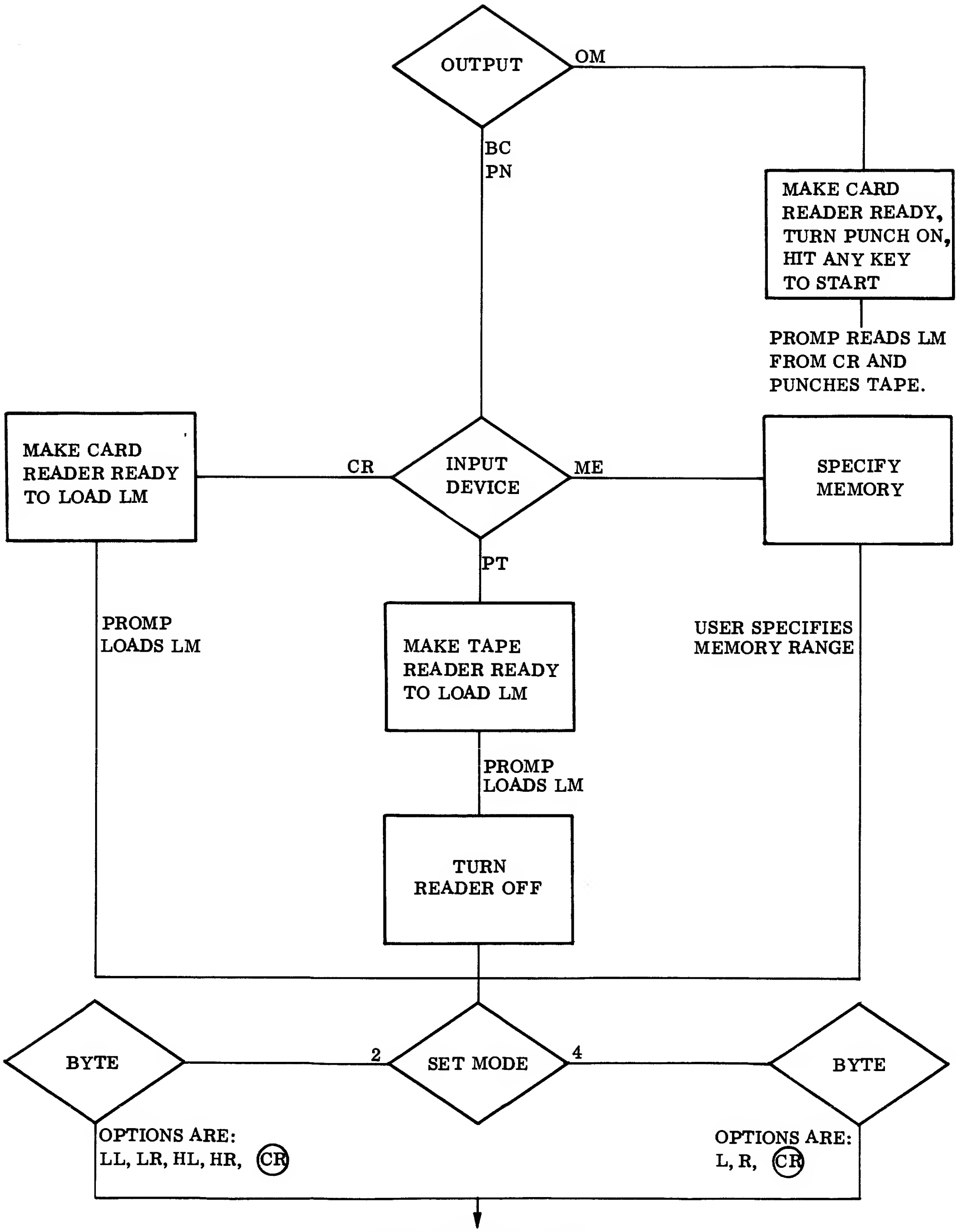


Figure 4-1. Operation Of The PROMP Program

CHANGE NOTICE NUMBER 2
November 15, 1974

Publication Number 4200025B
Order Number IMP-16S/025YB

IMP-16 Utilities
Reference Manual

This change is effective immediately.

Page 5-1. Section 4.2. Add to first paragraph:

"Those commands that require text mode processing are KB, RT, RC, MD, and MS."

Page 5-2. In first line, replace "Specifies" with "Separates".

Page 5-2. Section 5.3.3. Change last sentence of paragraph to read:

"The edit buffer is located in the remainder of memory above the EDIT16 code so its size is dependent on the machine memory space."

Page 5-3. Section 5.4.2. After the first paragraph, change to read as follows:

Example:

? RT 2 CR TURN READER OFF NOW	Apend two lines to the edit buffer. This message is typed after the second line is read.

"If the edit buffer becomes full when entering lines using KB, RT, and RC commands, the message

BUFFER FULL

is typed and EDIT16 goes into command mode. Again, an incomplete line is not entered to the buffer."

"If a BUFFER FULL message is typed, the user may recover by punching a few lines from the beginning of the buffer onto paper tape and, then, deleting those lines from the buffer."

Page 5-4. Section 5.4.8. Add to first paragraph:

"EDIT16 will type the line to be modified and, then, will prompt for modifications by typing ALTERS?."

Page 5-4. Section 5.4.8. Add after first paragraph:

"Typing CR as the first character after ALTERS? causes EDIT16 to terminate the modification and to prompt for a command."

Page 5-5. Add after sentence following second example of Section 5.4.8:

"If CTRL/Z is followed by another control character, EDIT16 will reprompt for ALTERS?."

Page 5-5. After the third example of Section 5.4.8 between ALTERS? and "Typing a CRTL/Q aborts the current line modification.", add the following:

"Attempting to add a line feed (LF) character will cause loss of a character from the original text."

"EDIT16 limits the maximum line length to 65 characters. If the user attempts to exceed this limit by inserting characters, the insertion will be aborted. The user, then, may delete characters from the end of the line, re-perform the insertion, and then add the deleted characters to the next line or insert them following the current line."

Page 5-6. Section 5.4.9. Immediately preceding 5.4.10 (that is, as the last of Section 5.4.9), add the following:

"The user will be prompted with ALTERS? for each line containing the specified string. If alteration of a particular line is not necessary, he may respond with a CR immediately following the prompt."

"If it is desired to terminate modifications, the user may type CTRL/Q."

Page 5-7. Section 5.4.15. Add new paragraph:

"When entering data via EDIT16, CTRL/I should be used to skip to the next tab setting."

Page 5-8. Add the following as Section 5.5.3:

"5.5.3 Restarting EDIT16

EDIT16 may be restarted by pressing INITIALIZE, setting the PC to the value contained in the data item RINIT1 (see listing), and pressing RUN."

Page C-1. FORMAT OF INSTRUCTIONS delete the words "to Memory" for second format, now "Register to Memory."

CHANGE NOTICE NUMBER 3
(28 February 1975)

Publication No. 4200025B
Order No. IMP-16S/025YB

IMP-16 Utilities
Reference Manual

This change is effective immediately for the following program:

PROMP 4300308C

Only changes incurred by Change Notice Number 1 are affected by this change notice. Thus, references to page numbers and sections apply to Change Notice Number 1. Changes or additions are underscored.

NOTE

Information in Change Notice Number 1 pertaining to other programs listed therein is still in effect.

Page 2 — affecting manual Section 4.1, addition of paragraph 2.3 should state:

If the BC tape option is selected, PROMP will calculate a 32-bit checksum of the tape and punch it at the end of the tape as eight hexadecimal characters in ANSI code.

Page 3 — affecting replacement of Section 4.2.3, Program Messages, of the manual, add second sentence so fourth paragraph states:

At this point, RLM records are converted from card to paper tape. GENLDR commands such as !RLM are ignored. After processing the END record, PROMP goes into a wait state. The user may now turn off the Teletype punch. Typing any key returns the PROMP program to the initial state.

CHANGE NOTICE NUMBER 4
(1 July 1975)

Publication No. 4200025B
Order No. IMP-16S/025YB

IMP-16 Utilities
Reference Manual

This change notice affects the IMP-16 General Loader Program, GENLDR.

Replace the existing section 2.5 (through 2.5.19) of the subject manual with the attached section 2.5.

2.5 GENLDR (IMP-16 GENERAL LOADER)

GENLDR is a stand-alone IMP-16 program that reads one or more load modules (LMs) produced by the IMP-16 assembler, performs relocation, resolves external linkages, and loads the LMs into main memory for execution. The load modules produced by the assembler may be read from either cards or paper tape.

The paragraphs that follow describe the commands to control GENLDR and the input sequences required to load an executable program into the IMP-16 main memory. Error messages and diagnostic output of GENLDR are also described.

NOTE

The prefix X' designates that the number that follows is hexadecimal.

2.5.1 Usage

GENLDR is a relocatable stand-alone IMP-16 program that may be loaded into memory by one of two absolute loaders: (1) ABSCR allows GENLDR to be input from cards and (2) ABSPT, from paper tape. Once loaded, GENLDR can accept input from either cards or paper tape; although, initially, it accepts input from the device from which it was loaded.

GENLDR occupies approximately 1568 words of memory and is typically loaded into upper memory. Programs cannot be loaded by GENLDR into memory that it occupies or uses for the symbol table it generates. However, GENLDR allows the user full use of base page. The memory layout is described in figure 2-3.

GENLDR is assembled relocatable and occupies the (relative) addresses X'09E0 through X'0FFF. It is recommended that it be loaded into the highest locations of memory available in the system, for ease of use and to enable GENLDR to remain resident while other programs are running. To load it into the top of an 8K memory, for example, first load it with the absolute loader, and then use GENLDR to relocate itself with the !OTS 1000 command. GENLDR would then reside in the memory locations X'19E0 through X'1FFF.

GENLDR is self-initializing; it may be entered at its entry point at any time. The entry point is X'0A5F in a 4K system. The value of AC3 is used to determine the initial load device: if AC3 = 0, the device is the Teletype: otherwise, it is the card reader. If GENLDR is entered after pressing initialize, the device will be the Teletype.

The IMP-16 assembler allows the user to allocate portions of his program in three ways:

- At an absolute memory location
- Relative to the origin of the base sector
- Relative to the origin of the top sector

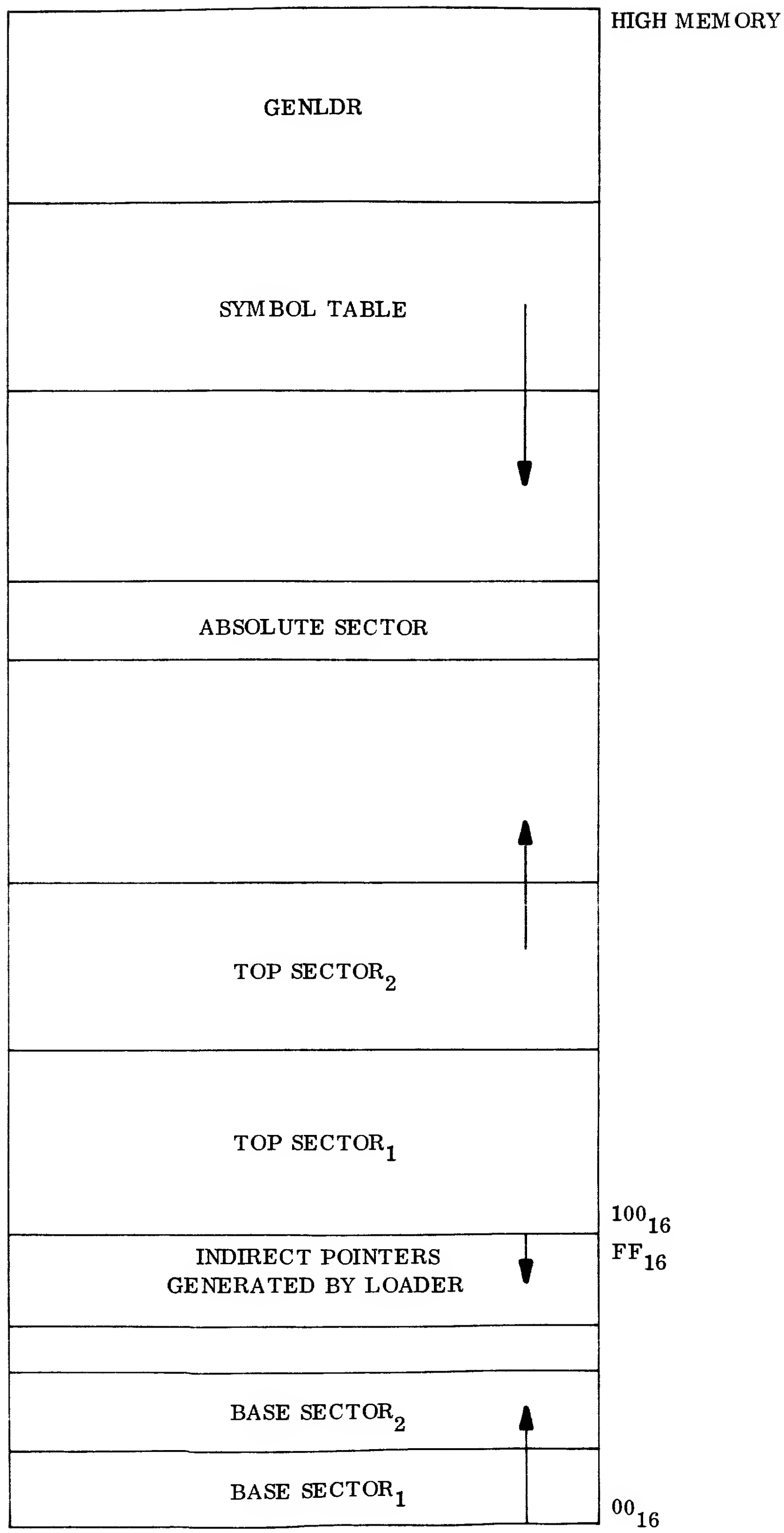


Figure 2-3. Memory Map

Typically, absolute allocation is employed to assign locations dependent upon equipment (for example, interrupt entrance address) or to communicate with special-purpose routines. The base sector must be located such that it is contained within the first 256 (X'100) locations of memory and typically contains data and pointers necessary for inter-LM communication. The top sector may reside anywhere in memory (subject to the limitations mentioned above) and normally contains the main portion of the LM. Care must be exercised to ensure that an absolute sector does not overlay a previously loaded base sector or top sector. (See !OBS and !OTS commands in the following paragraphs.)

Two other limitations are imposed upon the base sector by the IMP-16 computer architecture and the method for resolving external linkages. First, any base sector variable that is referenced by an indexed instruction must be allocated to one of the first X'7F locations of memory. Second, in resolving certain external linkages, GENLDR may force an indirect reference to a global variable through a pointer in the memory area X'FF and downward.

The area of IMP-16L memory between locations X'100 and X'11F is used by the control panel service routine and may not be used by the user. Above address X'FF, loading is limited only within the area occupied by GENLDR and the symbol table it generates. In the IMP-16P, GENLDR allows the locations used for its IMP-16L Teletype input/output to be overwritten, so the effective starting location of GENLDR is X'0A31. The GENLDR area may be used by the loaded program, after it receives control from GENLDR; but it is recommended that GENLDR be left in memory so that it may be reused without necessitating reloading.

As an entry point, GENLDR selects the last nonzero value specified for the set of LMs loaded. The entry point for any particular LM, if specified, appears in the end record of that LM. If the user desires, he may override the entry point selected by GENLDR by specifying the desired entry point in the !GO command (paragraph 2.5.4.13). If neither of these methods is chosen, GENLDR prints an "ENT?" error message and prompts for a new command.

2.5.2 GENLDR Input

GENLDR is controlled by commands and by the load module data. GENLDR reads commands and LMs from either cards or paper tape, and commands are available to switch between input devices. (See !CR and !TTY, paragraphs 2.5.4.6 and 2.5.4.7.) GENLDR does not recognize any distinction between the Teletype paper tape reader and the Teletype keyboard; therefore, the user may type in his commands at the keyboard and input the LM from paper tape. Commands entered on paper tape, the keyboard, or from the card reader are echoed back to the Teletype printer; the LM itself is not echoed.

Commands entered on punched cards must contain an exclamation point (!) in column 1. When the command is entered from the Teletype, GENLDR types the exclamation point to prompt for a command. Any additional exclamation point typed from the keyboard is ignored. If it is necessary to return to command mode while in the middle of an LM, an exclamation point may be typed on the keyboard (between LM records only), and GENLDR then prompts for a command. Any command may be given at this point, but only commands not affecting loading may be used if the LM is to be continued.

Input lines on the Teletype are terminated by a carriage return. A maximum of 72 characters is allowed in one Teletype input record; excess characters are not allowed. Only the characters from X'20 through X'5F are allowed in Teletype commands; any other character except carriage return causes the command to be aborted and a new prompt issued. The Null (X'00) and Rubout (X'7F) characters are, however, ignored.

2.5.3 GENLDR Output

GENLDR prints information descriptive of the loading process. The title information, absolute sector limits, base sector limits, top sector limits, and entry point address of each LM are printed on the Teletype. Unless !NLM is in effect, GENLDR types the following information for each LM:

```
MNEMONIC     STRING
  AAAA BBBB
  AS = XXXX:XXXX   BS = XXXX:XXXX  TS = XXXX:XXXX  ENT =XXXX
```

Where:

MNEMONIC	is the name of the LM from the title record.
STRING	is the qualifying string from the title record.
AAAA BBBB	are the LM source and object checksums, respectively.
AS = XXXX:XXXX	specifies the low and high addresses of the absolute sector (if any).
BS = XXXX:XXXX	specifies the base sector origin and last base sector address (if any).
TS = XXXX:XXXX	specifies the top sector origin and last top sector address (if any).
ENT =XXXX	is the entry address from the end record (if any).

All numbers (XXXX) are printed in hexadecimal notation.

If !SY or !ER is executed, GENLDR prints symbols as follows:

SYMBOL XXXX F

Where:

- SYMBOL is the symbol name.
- XXXX is the hexadecimal address of the symbol.
- F is one of the following:
  - M — multiply-defined symbol
  - U — undefined symbol
  - blank — defined symbol

  The address printed for an undefined symbol is the last address where the symbol is referenced (in a .WORD) or the base page pointer location.

If the !RA command is given to list the range of loaded addresses, or upon execution of a !GO command, the following information is typed:

AS = XXXX:XXXX BS = XXXX:XXXX TS = XXXX:XXXX PTR = XXXX:XXXX ENT = XXXX

Where the first hexadecimal value is the lowest address actually loaded in the specified sector, and the second hexadecimal value is the highest address actually loaded. PTR = XXXX:XXXX gives the limits of the loader-generated indirect pointers. These ranges are global; that is, they cover all programs loaded and are not affected by the use of the !LM or !NLM commands.

2.5.4 GENLDR Commands

All commands must begin with the exclamation point (!) in column 1 of the input record and the command string beginning in column 2. All commands have the format:

!CCC XXXX Comments......

Where CCC is the command name, of which only the first two characters are significant. XXXX is a hexadecimal value that must be separated from the command by at least one blank. Unless otherwise specified, where the term <hex-value> is used below, it represents a hexadecimal number in the range 0000 to FFFF. Leading zeros need not be specified, and only the last four significant digits are retained. If no value is specified, zero is used. Scanning of the operand is terminated by either the end of the line or encountering a character not in the hexadecimal set. Therefore, all commands may be commented if desired.

2.5.4.1 !OBS — Origin Base Sector

The origin for the next base sector is set to <hex-value>. If this command is not specified, the next base sector is loaded immediately following the previous base sector (the initial base sector starts at X'10 if no !OBS command is given). This command should be used to prevent loading a base sector on top of an absolute sector.

<Hex-value> must be in the range 0 ⩽ <hex-value> ⩽ X'FF or in the range X'FF01 ⩽ <hex-value> ⩽ X'FFFF. If the value is outside of these ranges, the base sector overflow message is given, the command is ignored, and GENLDR prompts for a new command from the Teletype.

2.5.4.2 !OTS — Origin Top Sector

The origin for the next top sector is set to <hex-value>. If this command is not specified, the next top sector is loaded immediately following the previous top sector. This command should be used to prevent loading a top sector on top of an absolute sector.

The highest value of <hex-value> is a function of the memory available, and must not cause overlaying of the locations occupied by GENLDR. If this command is not given, the first top sector is loaded at location X'120.

2.5.4.3 !RLM — Read Load Module

This command may precede each LM to be loaded. The LM is loaded from the same device from which the !RLM command is entered. The !RLM command is not required if input is from the card reader, but it is necessary in the Teletype mode to start the reading of the paper tape.

2.5.4.4 !CLR — Clear Memory and Restart GENLDR

All computer memory outside of GENLDR is cleared to zeroes; then, GENLDR is reinitialized and started from its entry point. Since this command clears any previously loaded information, it should only be used as the first command to GENLDR or as a means of reinitialization following an error.

2.5.4.5 !RE — Restart GENLDR

GENLDR is reinitialized and restarted at its entry point. This command does not clear memory, but is otherwise the same as the !CLR command, above.

2.5.4.6 !CR — Read Input from the Card Reader

Subsequent input is accepted from the card reader.

2.5.4.7 !TTY — Read Input from the Teletype

Subsequent input is accepted from the Teletype. The card reader should be turned off before using the Teletype, since the IMP-16P card reader interface is also used by the high speed paper tape reader. Teletype input is accepted from either the paper tape or the keyboard, but only commands are echoed to the Teletype printer.

2.5.4.8 !SY — Print the Symbol Table

The symbol table is printed upon execution of this command.

2.5.4.9 !ER — Print Symbols in Error

Multiply-defined and undefined symbols are printed when this command is read.

2.5.4.10 !RA — Type Global Loaded Range

The range of loaded addresses for each sector, plus the range of pointers generated (if any) and the current value of the entry point, is typed. This information is the same as printed on execution of a !GO command. This command may be used to determine the status of the loaded program if an error occurs.

2.5.4.11 !NLM — Inhibit Load Module Limit Printing

This command inhibits the printing of the title record information, checksums, and address ranges that are printed for each load module. Printing may be restored by the !LM command.

2.5.4.12 !LM — Enable Load Module Limit Printing

This command enables the printing of the information for each load module that was inhibited by !NLM. Printing of the information is the default state that is set upon initialization of GENLDR.

2.5.4.13 !GO — Execute the Loaded Program

The entry Point specified in the last LM loaded can be overridden by specifying the entry point address (<hex-value>). If <hex-value> is omitted, the last nonzero entry point specified is executed. If no nonzero entry point is specified and no value appears on the command, GENLDR prints an error message and prompts for a new command from the Teletype. Before transfer to the entry point, the global loaded program limits are printed on the Teletype.

2.5.5 Messages

The following messages may be output by GENLDR:

GENERAL LOADER (REV. X) READY — GENLDR is initialized and ready to read commands. X is the current revision level of GENLDR.

CMND? — The command is invalid or unrecognized. GENLDR prompts for a new command from the Teletype.

CHAR — An LM card contains characters other than 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Since invalid punch codes are translated as '?', this error can also occur on an invalid punch. Correct the card, reload it, and press RUN.

CKSM — Checksum error on the last record read. If the checksum field is 0000, no checksum test is made. Processing of the record is completed after 'RUN' is pressed. The read may be retried: reload the record and then press RUN.

BSOV — Base sector overflow. The run must be restarted, but the error may be corrected by proper use of the !OBS and !OTS commands.

TSOV — Top sector overflow. The run must be restarted, but the error may be corrected by proper use of the !OBS and !OTS commands. This error is caused by exceeding the upper limit of memory, X'FFFF.

AREA — Illegal attempt to load on top of the loader. Restart with valid !OBS or !OTS commands.

<address> MEM — Memory error at <address>: probably an attempt to load into non-existent memory.

ENT? — No entry point specified for program. GENLDR transfers control to the Teletype for a new command.

LENGTH — Record length error: the record is too long to fit in the buffer in GENLDR. The run may be restarted after the record is corrected.

EXTN — Unable to locate external symbol in Symbol Table. This error may be caused by attempting to load an LM with some missing symbol records or an erroneous patch which looks as if it is referencing an illegal external reference number. The run must be restarted.

ADDR — Addressing error. This error occurs under the following conditions and the run must be restarted:

1. Attempting to reference an indirect pointer generated by the assembler which, because of relocation, is forced to an address greater than X'7F.
2. Using an index register in an instruction referencing a base sector variable allocated in a memory address in the range X'80 to X'FF.
3. Attempting to use an index register in an instruction referencing an undefined external variable.
4. Referencing an undefined external variable in an instruction which either is flagged indirect already or cannot be so flagged.

SEQ — Record sequence error. The correct sequence is: (1) Title record, (2) Zero or more symbol records, (3) Zero or more data records, and (4) End record. Correct the record sequence, reload the LM, and restart the load.

SYST — System error caused by a malfunction in system software or hardware. Recovery is not possible; GENLDR should be reloaded.

SYMB — Symbol table overflow. Too many external symbols defined. The symbol table is allocated downward from the start of GENLDR; overflow will occur if an attempt is made to expand the symbol table into a region already loaded.

National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051
(408) 732-5000
TWX: 910-339-9240

National Semiconductor Electronics SDNBHD
Batu Berendam
Free Trade Zone
Malacca, Malaysia
Telephone: 5171
Telex: NSELECT 519 MALACCA (c/o Kuala Lumpur)

National Semiconductor GmbH
D 808 Fuerstenfeldbruck
Industriestrasse 10
West Germany
Telephone: (08141) 1371
Telex: 27649

National Semiconductor (UK) Ltd.
Larkfield Industrial Estates
Greenock, Scotland
Telephone: (0475) 33251
Telex: 778 632

NS Electronics (PTE) Ltd.
No. 1100 Lower Delta Rd.
Singapore 3
Telephone: 630011
Telex: 21402

REGIONAL AND DISTRICT SALES OFFICES

ALABAMA
DIXIE DISTRICT OFFICE
3322 Memorial Parkway, S.W. #67
Huntsville, Alabama 35802
(205) 881-0622
TWX: 810-726-2207

ARIZONA
*ROCKY MOUNTAIN REGIONAL OFFICE
7349 Sixth Avenue
Scottsdale, Arizona 85251
(602) 945-8473
TWX: 910-950-1195

CALIFORNIA
*NORTH-WEST REGIONAL OFFICE
2680 Bayshore Frontage Road, Suite 112
Mountain View, California 94043
(415) 961-4740
TWX: 910-379-6432

NATIONAL SEMICONDUCTOR
*DISTRICT SALES OFFICE
Valley Freeway Center Building
15300 Ventura Boulevard, Suite 305
Sherman Oaks, California 91403
(213) 783-8272
TWX: 910-495-1773

NATIONAL SEMICONDUCTOR
SOUTH-WEST REGIONAL OFFICE
17452 Irvine Boulevard, Suite M
Tustin, California 92680
(714) 832-8113
TWX: 910-595-1523

CONNECTICUT
AREA OFFICE
Commerce Park
Danbury, Connecticut 06810
(203) 744-2350

*DISTRICT SALES OFFICE
25 Sylvan Road South
Westport, Connecticut 06880
(203) 226-6833

FLORIDA
*AREA SALES OFFICE
2721 South Bayshore Drive, Suite 121
Miami, Florida 33133
(305) 446-8309
TWX: 810-848-9725

CARIBBEAN REGIONAL SALES OFFICE
P.O. Box 6335
Clearwater, Florida 33518
(813) 441-3504

ILLINOIS
NATIONAL SEMICONDUCTOR
WEST-CENTRAL REGIONAL OFFICE
800 E. Northwest Highway, Suite 203
Mt. Prospect, Illinois 60056
(312) 394-8040
TWX: 910-689-3346

INDIANA
NATIONAL SEMICONDUCTOR
NORTH-CENTRAL REGIONAL OFFICE
P.O. Box 40073
Indianapolis, Indiana 46240
(317) 255-5822

KANSAS
DISTRICT SALES OFFICE
13201 West 82nd Street
Lenexa, Kansas 66215
(816) 358-8102

MARYLAND
CAPITAL REGIONAL SALES OFFICE
300 Hospital Drive, No. 232
Glen Burnie, Maryland 21061
(301) 760-5220
TWX: 710-861-0519

MASSACHUSETTS
*NORTH-EAST REGIONAL OFFICE
No. 3 New England, Exec. Office Park
Burlington, Massachusetts 01803
(617) 273-1350
TWX: 710-332-0166

MICHIGAN
*DISTRICT SALES OFFICE
23629 Liberty Street
Farmington, Michigan 48024
(313) 477-0400

MINNESOTA
DISTRICT SALES OFFICE
8053 Bloomington Freeway, Suite 101
Minneapolis, Minnesota 55420
(612) 888-3060
Telex: 290766

NEW JERSEY/NEW YORK CITY
MID-ATLANTIC REGIONAL OFFICE
301 Sylvan Avenue
Englewood Cliffs, New Jersey 07632
(201) 871-4410
TWX: 710-991-9734

NEW YORK (UPSTATE)
CAN-AM REGIONAL SALES OFFICE
104 Pickard Drive
Syracuse, New York 13211
(315) 455-5858

OHIO/PENNSYLVANIA/ W. VIRGINIA/KENTUCKY
EAST-CENTRAL REGIONAL OFFICE
Financial South Building
5335 Far Hills, Suite 214
Dayton, Ohio 45429
(513) 434-0097

TEXAS
*SOUTH-CENTRAL REGIONAL OFFICE
5925 Forest Lane, Suite 205
Dallas, Texas 75230
(214) 233-6801
TWX: 910-860-5091

WASHINGTON
DISTRICT OFFICE
300 120th Avenue N.E.
Building 2, Suite 205
Bellevue, Washington 98005
(206) 454-4600

INTERNATIONAL SALES OFFICES

AUSTRALIA
NS ELECTRONICS PTY. LTD.
Cnr. Stud Road & Mountain Highway
Bayswater, Victoria 3153
Australia
Telephone: 729-6333
Telex: 32096

CANADA
*NATIONAL SEMICONDUCTOR CORP.
1111 Finch Avenue West
Downsview, Ontario, Canada
(416) 635-9880
TWX: 610-492-1334

DENMARK
NATIONAL SEMICONDUCTOR
SCANDINAVIA
Vordingborggade 22
2100 Copenhagen
Denmark
Telephone: (01) 92-OBRO-5610
Telex: DK 6827 MAGNA

ENGLAND
NATIONAL SEMICONDUCTOR (UK) LTD.
The Precinct
Broxbourne, Hertfordshire
England
Telephone: Hoddesdon 69571
Telex: 267-204

FRANCE
NATIONAL SEMICONDUCTOR
FRANCE S.A.R.L.
28, Rue de la Redoute
92260-Fontenay-Aux-Roses
Telephone: 660-81-40
TWX: NSF 25956F

HONG KONG
*NATIONAL SEMICONDUCTOR
HONG KONG LTD.
9 Lai Yip Street
Kwun Tung, Kowloon
Hong Kong
Telephone: 3-458888
Telex: HX3866

JAPAN
*NATIONAL SEMICONDUCTOR JAPAN
Nakazawa Building
1-19 Yotsuya, Shinjuku-Ku
Tokyo, Japan 160
Telephone: 03-359-4571
Telex: J 28592

SWEDEN
NATIONAL SEMICONDUCTOR SWEDEN
Sikvagen 17
13500 Tyreso
Stockholm
Sweden
Telephone: (08) 712-04-80

WEST GERMANY
*NATIONAL SEMICONDUCTOR GMBH
8000 Munchen 81
Cosimstrasse 4
Telephone: (0811) 915-027

***Microprocessor System Specialist Available**

D10M44